MASTER IN COMPUTER SCIENCE

Adaptation automatique de la qualité d'un streaming vidéo par les protocoles RTP/RTCP/RTSP

Henkes, Ronny

Award date:
2010

Awarding institution:
University of Namur

Facultés Universitaires Notre-Dame de la Paix, Namur
Faculté d'informatique
Année Académique 2009-2010

Adaptation automatique de la qualité d'un streaming vidéo par les protocoles RTP/RTCP/RTSP

Ronny Henkes

Mémoire présenté en vue de l'obtention du grade de Master en informatique

Résumé

Lors du téléchargement continu d'une vidéo d'un serveur vers un terminal client portable, on peut être confronté à une diminution de la bande passante due à l'utilisation d'un réseau mobile. Cette diminution de la bande passante peut dégrader la qualité de la vidéo téléchargée en continu. Le présent mémoire, va étudier les possibilités pour adapter la qualité d'un streaming vidéo utilisant les protocoles RTP/RTCP/RTSP, sur base d'un feed-back d'une mesure de la qualité vidéo, qui sera réalisée à l'aide d'un modèle VQM.
On propose une solution applicable à la partie client d'un système d'adaptation automatique de la qualité d'un service de streaming vidéo, sur base de la recherche effectuée, et on va l'implémenter sur un client en utilisant le framework GStreamer.

Mots Clés: VQM, feed-back, MOS, RTP, RTSP, RTCP, streaming video, adaptation qualité vidéo, GStreamer

Abstract

While streaming video content from a server to a mobile client terminal, we may be facing a reduction in bandwidth due to the use of a mobile network. This reduction in bandwidth may reduce the quality of the video. This thesis here will explore possibilities for adapting the quality of the streaming video, based on a feedback of a video quality metric, calculated on a VQM model. The video streaming uses RTP / RTCP / RTSP protocols.
We propose a solution for the client part of the system for automatic adjustment of the quality of a video streaming service, based on the aforementioned study, and we will implement it on a client using the GStreamer framework.

Keywords: *VQM, MOS, feed-back, RTP, RTSP, RTCP, video streaming, video quality adaptation, GStreamer*

Avant-propos

En préambule à ce mémoire, je souhaitais adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire.

Je tiens tout particulièrement à remercier mon promoteur et directeur de mémoire, le Professeur Laurent Schumacher, ainsi que son assistant George Toma, qui se sont toujours montrés à l'écoute et très disponibles tout au long de la réalisation de ce mémoire.

En fin, je remercie tous mes proches et amis pour leurs encouragements et leur contribution tout au long de mes études.

Table des matières

Acronymes

3GPP	*3rd Generation Partnership Project*
ADSL	*Asymmetric Digital Subscriber Line*
AMMF	*Average Maximal Motion by Frame*
AVC	*Advanced Video Coding*
BR	*Bit Rate*
BT	*Block Type*
CC	*Content Classes*
CIF	*Commun Intermediate Format*
CODEC	Codeur Decodeur
CRC	*Cyclic Redundancy Check*
DCT	*Discrete Cosine Transform*
DMIPS	*Dhrystone MIPS*
DSIS	*Double Stimulus Impairement Scale*
DVB	*Digital Video Broadcasting*
fps	*Frames per second*
FR	*Frame Rate*
GoP	*Group of Pictures*
HD (TV)	*High Definition (Television)*
IANA	*Internet Assigned Numbers Authority*
IDR Frame	*Instantaneous Decoder Refresh Frame*
MB	Macrobloc
MIPS	*Million instructions per second*
MOS	*Mean Opinion Score*
MPEG	*Motion Picture Experts Group*
MPQM	*Moving Picture Quality Metric*
MV	*Motion Vectors*
NTIA	*National Telecommunications Information Administration*
NR	*Non Reference*
PES	*Packetized Elementary Stream*
PSNR	*Peak Signal-to-Noise Ratio*
QCIF	*Quarter Commun Intermediate Format*
QoE	*Quality of Experience*
QoS	*Quality of Service*

QVGA	*Quarter VGA*
PDA	*Personal Digital Assistant*
RLC	*Run Length Coding*
ROI	*Region of Interest*
RR (VQM)	*Reduced Reference*
RR (RTCP)	*Receiver Report*
SAMVIQ	*Subjective assessment methodology for video quality*
SD (TV)	*Standard Definition (Television)*
SDES	*Session Description Protocol Security for Media Streams*
SIF	*Standard Interchange Format*
SR	*Sender Report*
SSCQE	*Single Stimulus Continuous Quality Evaluation*
TCP	*Transmission Control Protocol*
TS	*Transport Stream*
TTL	*Time To Live*
UDP	*User Datagram Protocol*
URI	*Uniform Resource Identifier*
VCL	*Video Coding Layer*
VLC	*Variable Length Coding*
VOD	*Video On Demand*
VOIP	*Voice over IP*
VQM	*Video Quality Metric*
VSTQ	*Video Service Transmission Quality*
WLAN	*Wireless Local Area Network*
XR	*Extended Report*

1 Introduction

Ce mémoire s'inscrit dans le contexte du projet WALCOMO. Le projet WALCOMO concerne la création de contenu et le contrôle de qualité pour le téléchargement continu de vidéo sur mobile [WALCOMO].
Lors d'un téléchargement continu de vidéo (streaming vidéo) d'un serveur vers un terminal client portable (GSM, PDA, Portable Wlan,...), on peut être confronté à une diminution de la bande passante due à l'utilisation d'un réseau mobile (GPRS, UMTS, WLAN,…). Cette diminution de la bande passante peut résulter en une dégradation de l'image affichée (artefacts, blocage de l'image,...), et ainsi diminuer la qualité de la vidéo téléchargée en continu. L'expérience de l'utilisateur final en sera réduite.

Le mémoire présenté ici va étudier la question des mécanismes que l'on peut utiliser lors de l'apparition des dégradations, pour rétablir la qualité de la vidéo téléchargée en continu et améliorer l'expérience de l'utilisateur final.

Le champ d'application du mémoire se limitera à la partie client.
Ce mémoire poursuit plusieurs objectifs :

- étudier les possibilités pour adapter la qualité d'un streaming vidéo lors d'une diminution de la bande passante sur base d'un feed-back d'une mesure de la qualité vidéo.
- proposer la partie client d'un système d'adaptation automatique de la qualité d'un service de streaming vidéo sur base de la recherche effectuée.
- implémenter la partie client de ce système.

L'étude va prendre en compte les protocoles RTP/RTCP/RTSP pour le transfert et le contrôle du flux vidéo entre le serveur de streaming et le client, ainsi que pour le feed-back de la mesure de la qualité vidéo.
Pour la mesure de la qualité vidéo, on se limitera à l'utilisation des modèles VQM.
La Figure1 présente le schéma général d'un streaming vidéo par les protocoles RTP/RTCP/RTSP en utilisant un client terminal mobile avec un feed-back de la qualité vidéo.

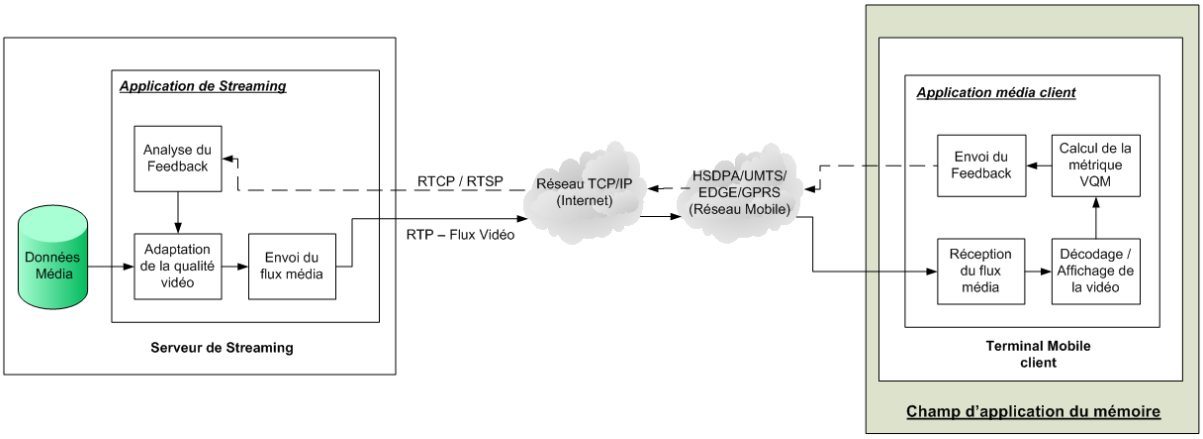


Figure 1 : Le streaming vidéo par les protocoles RTP/RTCP/RTSP.

Ce mémoire sera constitué de plusieurs parties :

-Déterminer les méthodes existantes pour la mesure de la qualité vidéo pour le streaming vidéo sur un terminal portable et évaluer si elles sont utilisables sur les terminaux.

- Analyser comment le feed-back de cette mesure peut être produit dans un format compatible avec la suite RTP/RTCP/RTSP.
- Proposer une solution et implémenter cette solution dans un client.

Bien que l'étude effectuée prenne en compte l'utilisation d'un terminal mobile comme client, l'implémentation est réalisée sur un ordinateur, vu qu'on ne dispose pas d'un terminal mobile adéquat.

Dans le chapitre deux, on va déterminer les spécificités de la vidéo mobile transmise par les protocoles RTP/RTCP/RTSP. Ces spécificités nous permettent de définir des critères de sélection pour le choix des métriques de qualité vidéo.

Le chapitre trois donne une introduction à l'évaluation de la qualité vidéo. Il traite de l'évaluation subjective de la qualité vidéo qui permet de déterminer un MOS (*Mean Opinion Score*) et de l'évaluation objective de la qualité vidéo, qui se base sur des modèles mathématiques. On va également identifier les dégradations de la qualité vidéo pertinentes pour le streaming vidéo mobile.

Le chapitre quatre présente un état de l'art de la mesure de la qualité d'une séquence vidéo. Les critères déterminés dans les chapitres précédents vont nous permettre de comparer et sélectionner le modèle VQM le mieux adapté à notre cas d'étude.

Dans le chapitre cinq, on va déterminer les possibilités pour renvoyer la métrique MOS calculée vers le serveur de streaming via les protocoles RTCP/RTSP. On décrit les possibilités des techniques standardisées ou non. On va alors choisir une technique sur base de critères de sélection déterminés.

Le chapitre six présente des techniques utilisées pour adapter la qualité vidéo pour le streaming. Les techniques discutées seront principalement localisées au niveau du serveur de streaming.

Le chapitre sept va présenter la solution générale pour réaliser la partie client d'un système d'adaptation automatique de la qualité d'un service de streaming vidéo en intégrant les différents composants. Il va également proposer une possibilité pour adapter le modèle VQM choisi à la vidéo avec un contenu continu.

Le chapitre huit propose une implémentation de la solution générale choisie dans le framework GStreamer. Deux parties principales seront traitées :

-l'implémentation du modèle VQM adapté dans un nouvel plugin GStreamer : NRVQM

-l'implémentation du feed-back MOS en adaptant les plugins GStreamer RTP/RTSP existants

Le chapitre neuf spécifie une série d'essais à effectuer pour tester la solution proposée implémentée dans GStraemer. On présente alors une analyse des résultats pour vérifier les fonctionnalités.

2 Spécificités de la vidéo mobile

L'objet de ce chapitre est de déterminer les spécificités de la vidéo mobile, c'est-à-dire déterminer les spécificités du streaming d'une vidéo d'un serveur vers un client mobile (téléphone mobile, Smartphone, PDA, …) en utilisant les protocoles RTP/RTCP/RTSP (Figure 2). Ce mémoire s'inscrit dans le contexte du projet WALCOMO. Le projet WALCOMO concerne la création de contenu et le contrôle de qualité pour le téléchargement continu de vidéo sur mobile [WALCOMO].

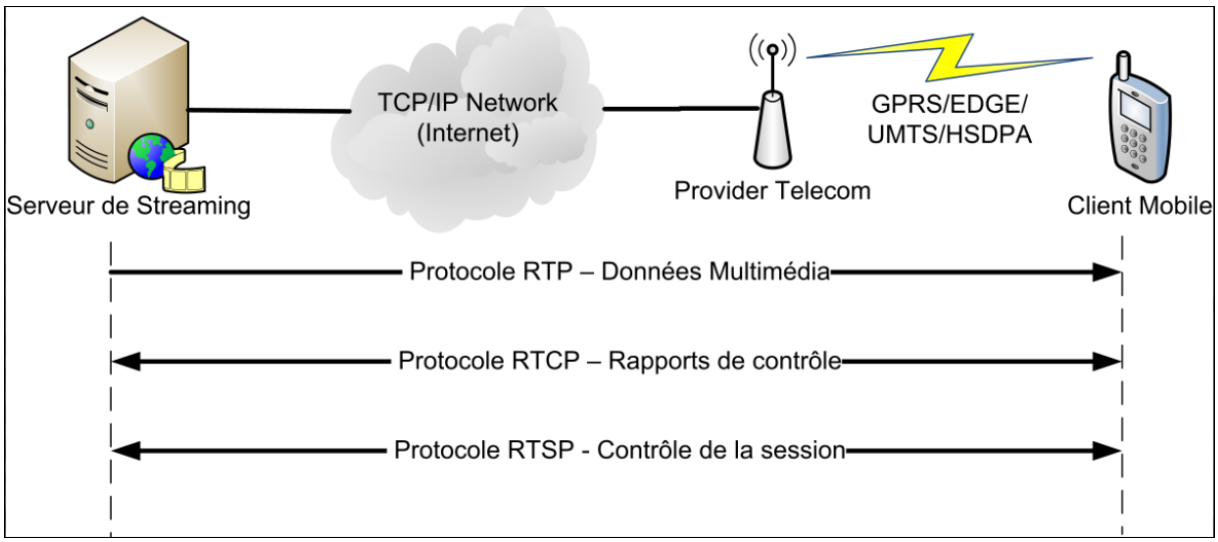


Figure 2 : schéma des streaming en vidéo mobile.

Pour pouvoir évaluer un contenu vidéo adapté pour des appareils mobiles il y a 2 contraintes principales à considérer :

- La bande passante réduite qu'offre un accès mobile
- La taille réduite des écrans des terminaux mobiles

On va également considérer les CODECS utilisés dans le streaming vidéo, la puissance des processeurs utilisés dans des terminaux mobiles ainsi que le protocole RTP.

2.1.La bande passante des accès mobiles

Le tableau 1 [BELGACOM2009] donne les bandes passantes pour l'accès mobile de Belgacom. Il faut noter que la technologie 3G de Belgacom dispose d'une couverture de 90% de la population (fin 2008) en Belgique et se concentre sur les agglomérations. (BASE ne propose la technologie 3G qu'à partir fin 2009, Mobistar dispose d'une couverture 3 G de 80 %, en 08/2009)

Une grande partie des appareils mobiles (entre autre l'Iphone de première génération) ne dispose pas encore de la technologie 3G et est donc limité à des vitesses de 56 kbps à 180 kbps pour l'accès des données (GPRS et EDGE).

Réseaux	GSM	GPRS	EDGE	UMTS	HSDPA	HSDPA
Vitesse*	9600 bps	56 Kpbs	180 Kbps	384 Kbps	3,6 Mpbs	7,2 Mbps
Date	1992	2000	2002	2004	2006	2008
Vitesse	2G	2,5G	2,75G	3G	3G Broadband	3,5 G
Type	modem	wap	data	vidéo	mobile internet	

(*) Vitesses théoriques de référence

Tableau 1 : Aperçu des réseaux data mobiles chez Belgacom. [BELGACOM2009]

Les vitesses sont des valeurs théoriques qui dépendent de la qualité et de l'utilisation de la connexion et de la disponibilité du service. En plus, vu qu'il s'agit d'une liaison radio, il y a une possibilité de perte de paquets de données ainsi que des erreurs de bit dues à des perturbations (CRC, correction des erreurs, demande de retransmission de paquets,…).

Donc on peut conclure que la bande passante dépend de :

- La technologie supportée par le terminal mobile
- L'opérateur télécom
- Le service disponible à un endroit et la qualité de connexion

Pour un appareil mobile compatible 3G+ (UMTS/HSDPA) avec un accès HSDPA la bande passante peut correspondre plus ou moins à un accès ADSL.
Pour un appareil mobile non compatible 3G ou ne disposant pas d'un accès UMTS/HSDPA la bande passante sera limitée à max. 180 kbps. (EDGE)

2.2.La résolution vidéo

Les terminaux mobiles ont généralement une taille d'écran et donc une résolution d'affichage petite. Des valeurs souvent rencontrées sont :

- Téléphone mobile : 128X220, 128X160 pixels
- Smartphone, PDA : 320X240 pixels

Le Tableau 2 donne des résolutions fréquemment utilisées pour le streaming vidéo mobile. [RIES2008] On utilise souvent une vitesse d'affichage de 5-15 images par seconde.

Nom		Résolution	Pour
QCIF	*Quarter Commun Intermediate Format*	176X144	Téléphone mobile
CIF	*Commun Intermediate Format*	352X288	Smartphone, PDA
SIF/QVGA	*Standard Interchange Format*	320X240	Smartphone, PDA

Tableau 2 : Formats de résolution utilisés fréquemment pour le streaming vidéo mobile.

2.3. Les CODECS

2.3.1 Généralités

On peut représenter les images d'une vidéo de plusieurs façons. On peut soit représenter chaque image en ces composantes rouges (R), vertes(G) et bleues(B), soit on peut représenter chaque image de la vidéo en un composant luminance (intensité lumineuse, Y) et chrominance (information de la couleur, U et V).
La luminance vaut :
Luminance= Y = 0.299 x R + 0.587 x G + 0.114 x B
La différence de bleu (U) est égal à :
U = 0.46 x (B-Y) / (1-0,114)
La différence de rouge vaut :
V = 0,615 x (R-Y) / (1-0,299)

Lors de la numérisation de la vidéo, les signaux de chrominance sont souvent échantillonnés avec une fréquence moindre que le signal de luminance. Un sous-échantillonnage utilisé est le 4 :2 :2, qui signifie que les signaux de chrominance (2 :2) sont échantillonnés avec une fréquence deux fois moindre que la luminance (4), la résolution horizontale de la chrominance est divisé par2. Il est souvent utilisé pour les formats vidéo de haute qualité.

Si on voulait transmettre une vidéo non compressée de résolution QCIF (176X 144), il nous faudrait une bande passante de 7,6 Mbps pour la partie vidéo, en effet :

Sampling vidéo 4 :2 :2 en 8 bits/Pixel :
176X144 (Luminance) + 2X88X72 (Chrominance U et V) par trame
Bande passante =8 X (176X144 + 2X88X72) X 25 = 7,6 Mbps pour 25 trames/s

Pour la résolution CIF (352X288) la bande passante passe à 30,4 Mbps. [HEYNA2003]

Un autre sous-échantillonnage souvent utilisé est le 4 :2 :0, qui signifie que la résolution horizontale et verticale de la chrominance est divisée par 2. Ce sous-échantillonnage est entre autre utilisé dans les standards MPEG.

Il est clair qu'il est impossible de faire du streaming vidéo vers des terminaux mobiles avec de la vidéo non compressée.
Il faut donc compresser la vidéo afin de pouvoir la transmettre. C'est le travail des CODEC (COdeur/DECodeur). Le tableau 3 montre plusieurs CODEC et leur utilisation.

CODEC	Utilisation fréquente	Bande passante	Résolution	Efficacité Compression
MPEG-1	CD-I, Video CD (VCD)	< 1,5 Mbps	352X288	<MPEG-2
MPEG-2	DVD, DVB, SAT digitale	< 80 Mbps(HD) <15 Mbps(SD)	SIF – HDTV	
MPEG-4 ASP	Streaming vidéo Internet, DIVX, Quicktime 6	5kbps – plusieurs Gbps	Sub-QCIF – HDTV (4kX4k)	>MPEG-2 <MPEG-4 AVC
WMV9 – VC-1	Windows Media Player,	96 kbps – 135 Mbps	QCIF - HDTV	2:1 MPEG-2
H.263	Vidéo UMTS Low bitrate, Real Video 8	n X 64kbps	Sub- QCIF - 16 QCIF	= MPEG-4 ASP
H.264 / MPEG-4 AVC	Streaming Vidéo Internet, mobile, Blue Ray Disc, Quicktime 7	5kbps – plusieurs Gbps	Sub-QCIF – HDTV (4kX4k)	2:1 MPEG-2

Tableau 3. Différents CODEC et leur utilisation. Sources [HEYNA2003], Wikipedia, [Microsoft2009]

Le CODEC H.264/MPEG-4 AVC est un des CODEC actuels les plus intéressants pour le streaming vidéo mobile vu son efficacité de compression (2 :1 par rapport au MPEG-2) et les possibilités de son utilisation (Sub-QCIF jusque HDTV, 5kbps jusque quelques Gbps).

2.3.2 Les CODEC de type MPEG

Plusieurs CODEC du Tableau 3, appartiennent à la famille des CODEC MPEG. On va donner quelques techniques employées par ces CODEC très utilisés, ce qui permettra de comprendre quelques problèmes qui peuvent apparaître au niveau de la qualité vidéo.

On travaille sur 2 axes pour compresser les données vidéo :

- L'axe temporel ou Inter-trame : compression entre plusieurs images
- L'axe spatial ou Intra-trame : compression dans une image

L'axe temporel inter-trame :

Si on prend un flux vidéo d'une vidéo non compressée QCIF à 25 fps, chaque image (I) de la vidéo est transmise complètement (ce qui donne une bande passante nécessaire de 7,6 Mbps).
Donc on transmet le flux suivant: I I I I I I I I I I I I … (I=Image Intra Trame)

Une idée du CODEC est de ne transférer que les changements d'une image à l'autre. Les images successives ne sont décrites que par la façon dont elles diffèrent par rapport à l'image précédente. Le décodeur ajoute cette différence aux images précédentes pour afficher l'image actuelle.

S'il y a beaucoup de mouvement, on utilise la compensation de mouvement. Pour cela, on divise l'image en Macroblocs, qui sont constitués de plusieurs blocs de luminance et de chrominance en tenant compte du sous-échantillonnage. Ainsi pour un sous-échantillonnage de 4 :2 :2, un Macrobloc est constitué de 8 blocs de taille 8X8 (16X16) Pixels (4 blocs Y, 2 blocs U et 2 blocs V). On évalue le mouvement de ces Macroblocs entre 2 images. Ceci va nous donner des vecteurs de mouvement, qui donnent le déplacement d'un Macrobloc d'une image par rapport à la suivante et des erreurs de prédiction qui permettent au décodeur de reconstruire l'image.
Donc en plus des Images I (intra Trame) il y a les trames P. La première trame P est codée en utilisant la trame I comme base pour déterminer les vecteurs de mouvement et les erreurs de prédiction. Les trames P suivantes utilisent la trame P précédente comme base.
Donc on aura le flux suivant : I P P P P P P I P P (P=différence par rapport à l'image précédente I ou à l'image précédente P)
L'ensemble des trames entre 2 trames I s'appelle GoP (Group of Pictures).
On peut avoir un troisième type de trame : La trame B. Elle est codée en utilisant la trame I ou P précédente et la trame I ou P suivante comme base, et en déterminant les vecteurs de mouvement et erreurs de prédiction entre ces trames.
Un exemple de GOP : I B1 B2 P1 B3 B4 P2 B5 B6 I2
Comme le décodeur doit connaître la trame P1 avant B1 pour pouvoir le calculer, l'ordre de transmission est le suivant : I P1 B1 B2 P2 B3 B4 I2 B5 B6 …
(Si lors de la transmission une Image I est perdue, les trames P et B suivantes ne pourront être calculées correctement et il faut attendre la trame I suivante, ce qui peut engendrer des dégradations à l'affichage)

On peut remarquer que pour le codage MPEG-4 AVC, un GOP peut contenir plusieurs trames I, et les trames P peuvent référencer des trames avant les trames I. C'est pourquoi en MPEG-4, on a introduit des trames IDR qui sont les premières trames d'un GOP. Une trame qui se situe après la trame IDR ne peut référencer une trame avant la trame IDR. On ne peut avoir qu'une seule trame IDR par GOP.

L'axe spatial intra-trame :

Lorsqu'on fait une analyse de fréquence spatiale d'une image, on peut constater que de petits objets sont représentés par des hautes fréquences et les objets larges par des basses fréquences. En plus, dans la plupart des zones de l'image, seulement quelques fréquences dominent et la plupart des autres sont absentes. Si on a une image avec un objet large uniforme, il n'y aura pas de hautes fréquences spatiales à l'intérieur de l'objet et celles-ci ne doivent donc pas être transmises [SNELL2009].
Dans le codeur MPEG on divise l'image en tableaux de 8X8 Pixels et on effectue une DCT (*Discrete Cosine Transform*) pour réaliser l'analyse de fréquence spatiale. Les tableaux sont alors transformés en tableau de coefficients. Chaque coefficient représente l'amplitude d'une fréquence spatiale présente.
On applique alors des matrices de quantification pondérées (préexistant) à ces tableaux qui vont quantifier les coefficients d'une façon pondérée (les hautes fréquences seront quantifiées de manière plus fines que les basses fréquences). Lors de cette opération beaucoup de coefficients peuvent être arrondis à 0 et donc ne doivent pas être transmis. (Ce qui conduit parfois à l'affichage des contours des Blocs 8X8 : c'est l'effet de bloc)

A la fin du processus on réalise un encodage par entropie (VLC ou RLC), qui va supprimer la redondance dans le flux de données. Le VLC utilise souvent le code de Huffmann.
La Figure 3 montre le processus d'encodage/décodage d'une image I. Le processus d'encodage de trames P et B est plus compliqué.

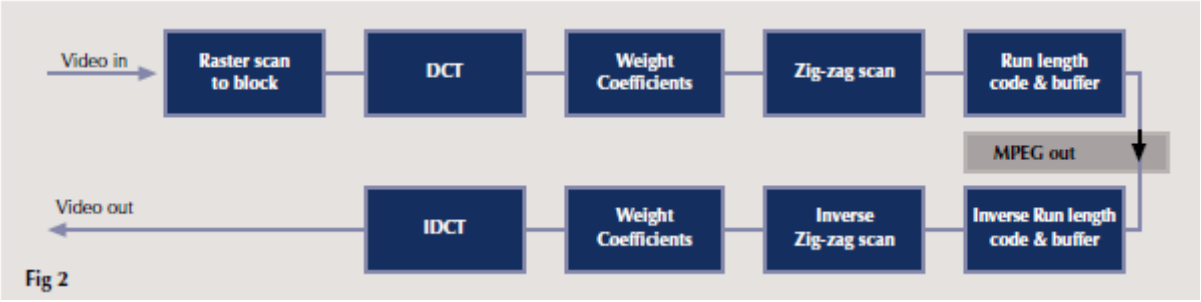


Figure 3 : Le processus d'encodage/décodage d'une trame-I [SNELL2009].

2.4.Les processeurs des terminaux mobiles

Les terminaux mobiles devront être capables de calculer les métriques de qualité vidéo en temps réel (en plus du décodage de la vidéo et de l'audio). Suivant le modèle du VQM choisi, ces calculs pourraient utiliser beaucoup de puissance CPU. Il est dès lors intéressant d'avoir une idée des performances des processeurs utilisés dans les terminaux mobiles (Smartphones, PDA, …).

Le Tableau 4 montre quelques spécifications de processeurs, dont l'architecture a été développée par la marque ARM, qui sont fréquemment utilisés dans des téléphones mobiles, Smartphones et PDA.

Famille	Core	Mips@MHZ typique	Utilisé dans
StongARM	SA-1110	233 MHz 1 DMIPS/MHz	HP Jornada 7xx (PDA 2003)
ARM9E	ARM926EJ-S	220 MIPS@200 MHz	Sony Ericsson K, W Series (Tél. mob.) K750 @ 112MHz (2005) Benq x65 (Tél. mob.) K320i@112 MHz(2009)
Xscale	PXA27x	800 MIPS@624MHz	Motorola Ezx Platform : A728, RokrE6 @312 MHz (Smartphone) Fujitsu Siemens LOOX N560 (PDA)
XScale	Monahans	1000MIPS@1.25GHz	Samsung Omnia (tél. mob., Smartphone)
ARM11	ARM1136J(F)-S	740MIPS@532-665 MHz	Nokia N95, N82 (Smartphone)
CORTEX	Cortex-A8	2 DMIPS/MHz 600-1000 MHz 1200-2000 DMIPS	iPhone 3GS (Smartphone) (2009) Nokia N900 (Internet Tablet) Palm Pre (Smartphone)

Tableau 4: Quelques spécifications de processeurs ARM [ARM2009].

On peut voir que les processeurs ont une performance de 1-2 Dhrystone MIPS par MHz suivant le type. Les terminaux les plus puissants sont les Smartphones de dernière génération (iPhone 3GS (600Mhz), Nokia N900 (600 MHz), Palm Pre (600 MHz)) avec une performance aux alentours de 1200 DMIPS. Un téléphone mobile d'entrée de gamme comme le K320i de Sony Ericsson à 112MHz arrive aux alentours de 120 DMIPS. Les Smartphones et PDA de gamme moyenne se situent souvent entre 450-900 DMIPS.
Il faut remarquer que les Smartphones peuvent contenir un accélérateur graphique/vidéo qui s'occupe du décodage de la vidéo.

Dans [BT2003], les auteurs présentent les résultats de leur implémentation du CODEC H.264 sur les appareils suivants : Smartphone Nokia 7650 (ARM9 à 100MHz, +/- 110 MIPS), PDA Compaq iPAQ (250 MHz, +/- 250MIPS) et sur un PC avec un processeur Intel P4 2 GHZ (+/- 4000 MIPS). Il s'agit d'une implémentation purement applicative qui n'utilise pas les possibilités d'accélération Hardware. Le Tableau 5 donne les résultats des performances de décodage en FPS pour le décodage d'une vidéo H.264 pour des résolutions différentes.

Dans [HOROWITZ et al, 2003] les auteurs ont également implémenté un décodeur H.264 Baseline Profile et ont effectués des mesures sur un PC Pentium III 600 (+/- 1700MIPS). Pour une vidéo CIF avec un débit binaire de 313 Kbit/s (15 FPS), ils ont mesuré une performance de décodage de 82 FPS. Pour une vidéo QCIF 64 Kbit/s, ils ont mesuré une performance de décodage de 330 FPS.

	QCIF	CIF	SD
Intel P4 2.0 GHz +/- 4000 MIPS	500 FPS	125 FPS	30 FPS
Intel P3 600 MHz +/- 1600 MIPS	330 FPS	85 FPS	Non testé
iPaq 250 MHz +/- 250 MIPS	15 FPS	/	/
Nokia 7650 100 MHz +/- 110 MIPS	12 FPS	/	/

Tableau 5 : Performance de décodage en fonction de la résolution [BT2003], [HOROWITZ et al, 2003]

Sur le site "http://www.smartphonemag.com/ cms/blogs/3/ the_h_264_a_k_a_mpeg_ 4_part_10_and_avc_b", l'auteur a effectué des tests avec le décodage H.264 sur des Smartphones. Il en résulte que des Smartphones de 400MHz (+/- 500 MIPS) peuvent décoder des vidéo en QVGA en temps réel.

Les performances de décodage dépendent largement de l'implémentation du CODEC H.264 et du débit binaire de la vidéo. Plus le débit binaire de la vidéo est haut, plus la performance de décodage est faible.

Ont peut en conclure :

- Les terminaux de faible puissance (moins de 400 MIPS) arrivent tout juste à décoder une vidéo en format QCIF. Ils ne sont pas capables de décoder les vidéos en CIF avec des valeurs FPS normales en temps réel.
- Les Smartphones et PDA de gamme moyenne (+/- 450-900 MIPS) peuvent décoder les vidéos QCIF et CIF (ou QVGA), mais les réserves de puissance de calcul pour le traitement du VQM sont limitées en CIF.
- Les terminaux de dernière génération (>1000 MIPS) peuvent décoder sans problème les vidéos QCIF et CIF et ont encore des réserves de puissance de calcul pour le traitement du VQM.

2.5.Le streaming vidéo avec le protocole RTP

Dans le contexte de ce mémoire, on utilise les protocoles RTP/RTCP/RTSP pour pouvoir effectuer le streaming d'une vidéo d'un serveur vers le client média. On va donner un aperçu du protocole RTP utilisé. Les protocoles RTCP/RSP seront traités au chapitre 5.

Le protocole RTP spécifié dans [RFC3550] propose des services pour le transport de bout-en-bout pour des applications envoyant des données orientées temps réel, entre autre des applications multimédia. Il permet entre autre l'identification du type de la charge (payload), la numérotation des séquences et le timestamping. Il est situé dans la couche transport du modèle TCP/IP et utilise surtout le protocole UDP pour le transport.

RTP en soi n'est pas complet, il a été conçu pour pouvoir être étendu. Pour cela, RTP se base en plus sur un document de spécification de profile et un document de spécification du format de charge. La spécification de profile concerne la définition d'un ensemble de codes de types de charge et leur correspondance vers leur format de charge ainsi que des extensions possibles. Le format de charge définit comment une charge est transportée dans RTP. La Figure 4 montre l'architecture du protocole RTP.

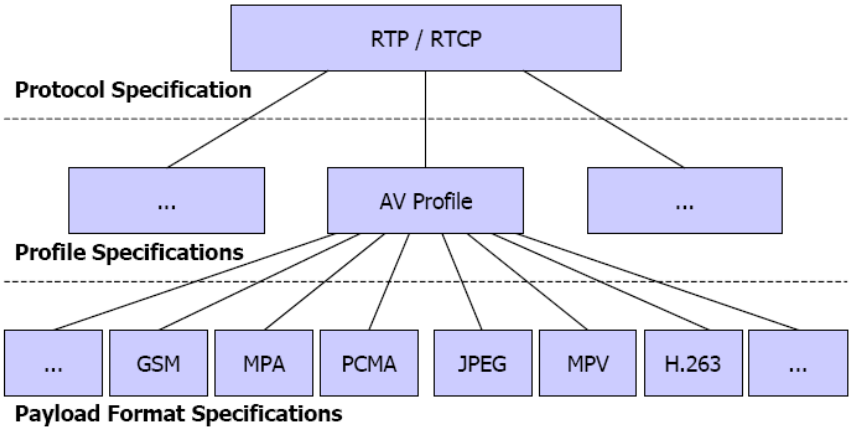


Figure 4 : l'architecture du protocole RTP [BOOK2000].

Le protocole RTP est souvent utilisé avec le RTP Control Protocol (RTCP), qui transmet des rapports d'informations entre les participants pour permettre le contrôle de la qualité de service. Le RTCP sera présenté dans un chapitre ultérieur.

Le RTP peut fonctionner en multicast, c'est-à-dire un serveur envoie des données vers plusieurs clients, ou en unicast, c'est-à-dire un serveur envoie des données vers un seul client.

RTP permet également l'utilisation des mixers, qui permettent de manipuler et combiner des paquets RTP d'une ou de plusieurs sources et d'acheminer le nouveau paquet vers la destination en adaptant les informations de timing et de source.
Les translators permettent également de manipuler des paquets RTP, mais ne changent pas l'information de source. Les translators sont utilisés comme convertisseurs d'encodage, ou des réplicateurs de multicast vers unicast par exemple.

Dans le contexte de ce mémoire, on se limite au cas d'un serveur de streaming (source) qui transmet le flux vidéo vers un client média. Ceci correspond à une utilisation du protocole RTP unicast avec une seule source.

2.6.Conclusion

Ce chapitre nous a permis de déterminer les spécificités du streaming de la vidéo sur mobile :

-bande passante variant de 180kbps à 7,6 Mbps
-perte de paquets et erreurs de bit possibles
-résolution vidéo QCIF/CIF/QVGA
-frame rate de 5-15 images par seconde
-codec de la famille MPEG (surtout H.263 et H.264)
-performance des terminaux mobiles souvent limitée

Ces spécificités peuvent entre autres être utilisées comme critères de sélection pour le choix des métriques de qualité vidéo.

3 Evaluation de la qualité vidéo

« La qualité vidéo est une caractéristique d'une vidéo qui est passée par un système de transmission/processing vidéo, une mesure formelle ou informelle de la dégradation perçue de la vidéo » (« *Video Quality* » sur Wikipedia 21/08/2009)

L'objectif de ce chapitre est de déterminer comment l'évaluation de la qualité vidéo peut être effectuée et de déterminer les dégradations de l'image vidéo pertinentes dans notre cas d'utilisation.

3.1 Evaluation subjective de la qualité vidéo

« La qualité vidéo subjective est une caractéristique subjective de la qualité vidéo. Elle concerne la manière dont une vidéo est perçue par un groupe de personnes et désigne son opinion pour une séquence vidéo particulière. » (« *Subjective video quality*» sur Wikipedia 21/08/2009)

L'idée est de faire évaluer une série de séquences vidéo par un groupe de personnes de façon standardisée. Les personnes devront donner une note à la fin de chaque séquence vidéo, qui dure normalement de 10-15 s. A la fin, on peut alors calculer une valeur moyenne pour chaque séquence, le *Mean Opinion Score* (MOS).

Il y a une série de standards qui ont été établis, qui décrivent des méthodes pour déterminer comment les séquences sont présentées aux personnes et comment leur opinion est collectée. Ces méthodes peuvent différer suivant le type de vidéo en question (Télévision, Vidéo multimédia,..) [FGIPTV2007]
L'ITU-R propose la recommandation BT.500 qui concerne principalement les images de télévision. Plusieurs méthodes pour la mesure de la qualité vidéo subjective sont proposées :

- DSIS : *Double Stimulus Impairement Scale* (La vidéo originale et la vidéo dégradée sont affichées en même temps)
- SSCQE : *Single Stimulus Continuous Quality Evaluation* (Seule la vidéo dégradée est affichée)

L'ITU-R BT.500-7 recommande l'échelle suivante sur 5 valeurs :

5 - imperceptible
4 – perceptible, mais pas gênant
3 – peu gênant
2 – gênant
1 – très gênant

L'ITU-T Rec P.910 concerne la vidéo multimédia :

- ARC : *Absolute Category Rating (Single stimulus)*

L'ITU-R WP6Q propose la méthode SAMVIQ (*Subjective Assessment Methodology for Video Quality*) qui est également orientée vers la vidéo multimédia [PASTRANA2006]. La méthode SAMVIQ donne des résultats comparables à la méthode ARC [Huynh-Thu2007].

3.2 Evaluation objective de la qualité vidéo

L'évaluation objective de la qualité vidéo consiste à utiliser un modèle mathématique qui se base sur des caractéristiques de la vidéo, qui peuvent être mesurées objectivement et calculées automatiquement par un programme. Cette métrique devra avoir une bonne corrélation avec le MOS de la qualité subjective de la vidéo.

Les modèles peuvent être classés dans trois catégories : [Winkler2008]

- Métriques de données
- Métriques de l'image
- Métriques basées sur le Packet- et Bitstream
- Métriques hybrides

3.2.1 Métriques de données

Ces métriques se basent sur les données de la vidéo décodée. Ils ne considèrent pas la perception humaine de la vidéo en question, mais se basent seulement sur des calculs entre des Pixels. [Winkler2008] les caractérise comme « distorsion-agnostic » et « content-agnostic ». Une métrique très utilisée est le PSNR (*Peak-Signal-to-Noise Ratio*). Bien que le PSNR soit très utilisé, sa relation avec la qualité vidéo perçue par des sujets n'est qu'approximative. Elle ne sera donc plus considérée dans la suite.

3.2.2 Métriques de l'image

Ces modèles se basent souvent sur la perception humaine de la vidéo décodée. Ils mesurent une ou plusieurs dégradations de la qualité vidéo (voir section 3.3) ou des caractéristiques et calculent un résultat de prédiction. Cette valeur de prédiction doit avoir une bonne corrélation avec le MOS. Ces modèles sont souvent complexes et nécessitent donc une puissance de calcul en conséquence.

3.2.3 Métriques basées sur le Packet- et Bitstream

Les métriques basées sur le Packet- et Bitstream se basent sur des paramètres extraits directement des paquets IP ou du bitstream encodé. Ces paramètres peuvent être extraits avec un décodage minimal et nécessitent une puissance CPU minimale. Ces métriques doivent être adaptées à des CODEC et protocoles réseaux différents. Ce sont des métriques de données, donc elles ne prennent pas en compte la perception humaine de la qualité vidéo.

3.2.4 Métriques hybrides

Les métriques hybrides se basent d'un côté sur des paramètres extraits des paquets et Bitstream, (qui sont facilement décodables) et d'un autre côté elles se basent sur des métriques de l'image qui ont une bonne corrélation avec le MOS.
Ainsi on essaye de limiter la puissance de calcul nécessaire de la métrique en gardant une bonne corrélation avec le MOS.

Le Tableau 6 donne une comparaison entre les différentes métriques.

Métrique	Image	Paquet-Bitstream	Hybride
Entrée	Signal vidéo décodée	Paramètres paquet-bitstream	Paramètres et signal
Avantage	Près du signal reçu par le client	Puissance CPU faible	Compromis
Inconvénient	Puissance CPU nécessaire élevée	Lien paramètre/effet pas explicite	Dépend de la technologie

Tableau 6 : Métriques de l'image vs Métriques paquet-Bitstream. [RAMIN2008] [Winkler2008]

3.2.5 Disponibilité de la vidéo de référence

Il est également possible de classer les modèles sur base de la disponibilité de la vidéo de référence. On distingue les trois catégories suivantes :

- *Full Reference* (FR) : on dispose de la vidéo originale et de la vidéo reçue pour calculer la qualité vidéo. On peut calculer la différence de qualité en comparant chaque pixel d'une trame de la vidéo originale avec le pixel correspondant de la trame de la vidéo-reçue. Cette approche n'est pas très bien adaptée pour le streaming vidéo puisqu'il faut disposer de la vidéo originale au récepteur ce qui n'est généralement pas le cas. L'avantage est qu'on peut calculer plus précisément la différence de qualité entre la vidéo originale et celle reçue. Ils

performent souvent mieux que les RR et NR –VQM. On appelle ces modèles des FR-VQM.

- *Reduced Reference* (RR) : on dispose d'informations partielles sur la vidéo originale et on dispose de la vidéo reçue pour calculer la qualité vidéo. On calcule des caractéristiques sur la vidéo originale, qu'on transmet en plus de la vidéo vers le récepteur. Celui-ci calcule ces caractéristiques sur la vidéo reçue et les compare avec les caractéristiques de la vidéo originale pour déterminer la qualité vidéo. Comme il faut transmettre des informations en plus de la vidéo vers le récepteur, on diminue la bande passante disponible pour la vidéo. Cela pourrait affecter la qualité vidéo lorsqu'on dispose d'une connexion à bande passante basse. Les modèles RR performent souvent mieux que les NR-VQM. On appelle ces modèles des RR-VQM.

- *Non reference* (NR): on ne dispose que de la vidéo reçue pour calculer la qualité vidéo. Cette approche est intéressante si on dispose d'une connexion à bande passante limitée. On appelle ces modèles *Non Reference Video Quality Metric* ou **NR-VQM.**

3.3 Dégradations de la qualité vidéo

Dans [ATIS2006] on retrouve une liste avec des définitions de dégradations de la qualité vidéo. Voici une sélection :

a. *Block Distortion* : distorsion de l'image caractérisée par l'apparence de la structure d'encodage en bloc. S'appelle également « *tiling* ».
b. *Blurring* (flou) : une distorsion globale sur toute l'image, caractérisée par une réduction de la netteté des bords et du détail spatial.
c. *Color Errors* : distorsion de toute l'image finale ou d'une partie, caractérisée par l'apparence de niveaux de saturation ou des couleurs non naturelles ou inattendues.
d. *Edge Business* : distorsion localisée sur ou près des bords des objets, et caractérisée par ses caractéristiques temporelles et spatiales.
e. *Temporal Edge Noise* : une forme d'edge business caractérisée par une netteté variable dans le temps des bords des objets (*shimmering*).
f. *Spatial Edge Noise* : une forme d'edge business caractérisée par une distorsion variable dans l'espace près des bords des objets.
g. *Mosquito Noise* : une forme d'edge business associée parfois avec le mouvement. Caractérisée par des artefacts se déplaçant autour des bords et/ou un bruit en forme de marbre superposé sur les objets.

h. *Error Blocks* : une forme de distorsion de bloc où un ou plusieurs blocs dans l'image n'ont pas de ressemblance avec la scène actuelle ou précédente et qui contrastent souvent avec les blocs adjacents.
i. *Jerkiness* (instable) : un mouvement qui était régulier et continu est perçu comme une série de '*snapshots'*(photo) distincts.
j. *Motion-Related Artifacts* (artefacts lié au mouvement) : distorsion du mouvement de la vidéo potentiellement observable. Parfois, cette distorsion devient plus observable s'il y a beaucoup de mouvement dans la vidéo. La distorsion peu apparaître comme distorsion de bloc, *jerkiness*, traînée ou comme une autre dégradation.
k. *Motion Response Degradation* : la détérioration du mouvement de la vidéo se présentant comme une perte de la résolution spatio-temporelle.
l. *Object Persistence* : distorsion où l'objet qui est apparu dans une image vidéo précédente (et qui ne doit plus être présent dans l'image actuelle) reste dans l'image actuelle et les images suivantes comme silhouette ou image estompée.
m. *Objet retention* : distorsion où une partie d'un objet qui est apparue dans une image précédente (et qui ne doit plus être présent dans l'image actuelle) reste dans l'image actuelle et les suivantes.
n. *Scene cut response* : dégradations perçues associées avec une coupure de scène.
o. *Smearing* (traînée) : distorsion localisée sur une sous-région de l'image reçue, caractérisée par une netteté réduite sur les bords et du détail spatial.

Dans [Huynh-Thu2006], les auteurs introduisent une autre dégradation : *jitter*. Lorsqu'il y perte de paquets, le décodeur ne peut pas calculer complètement l'image. Certains décodeurs répètent alors la dernière bonne image qu'ils avaient calculés, jusqu'à ce qu'ils reçoivent de nouveau une image complète. Les auteurs discutent également l'impact important des aspects temporels de la qualité vidéo (*jitter* et *jerkiness*) sur la qualité vidéo perçue.

Dans [WANG2002], les auteurs considèrent le *Block Distorsion* et *blurring* comme les artefacts principaux pour la compression JPEG (qui est une compression DCT).

Dans [RAMIN2008], les auteurs divisent les problèmes principaux en 2 axes :

- L'axe temporal: *jerky motion, freezes, fluidity break*. Ce qui correspond à *Jerkiness* dans [ATIS2006].
- L'axe spatial: artefacts (*blockiness, ringing*) et *blur-based* (flou). Ce qui correspond à *Block Distorsion* et *Blurring* dans [ATIS2006].

En résumé, les dégradations principales qui ont un effet sur la MOS sont d'un côté des dégradations sur l'axe spatial (dans l'image) (*Block distorsion,* flou*, ringning*), et d'un autre

côté des dégradations sur l'axe temporel (mouvement) (*Jitterness, jerkiness, fluidity break*,…).
Dans le contexte du streaming de la vidéo sur terminaux mobiles (perte de paquets, erreurs de bits,…) le *jitter/jerkiness* est une dégradation importante.

3.4 Conclusion

On a d'abord parlé de l'évaluation subjective de la qualité vidéo qui consiste à faire évaluer la vidéo par des personnes, qui donnent leur opinion subjective de la qualité perçue. Ceci permet d'établir le MOS.
Puis on abordé l'évaluation objective qui consiste à créer des modèles mathématiques à partir de caractéristiques mesurables et de métriques. Ils donnent une évaluation objective de la qualité vidéo. Pour valider un modèle, les résultats d'une évaluation objective devront avoir une bonne corrélation avec les résultats de l'évaluation subjective. Une possibilité de réaliser cela est de prendre en considération la perception humaine de la vidéo décodée et de mesurer les dégradations de la qualité vidéo. (Métrique de l'image)
Dans le contexte du streaming vidéo considéré ici, on ne dispose pas de la vidéo de référence, donc une métrique NR (*Non Referenced*) devra être utilisée.
Finalement on a abordé les dégradations de la qualité vidéo qui peuvent se présenter.
Le "*jitter/jerkiness*" est une dégradation dont il faut tenir compte, lorsqu'il y a perte de paquets et a une grande influence sur le MOS.

4 Modèles VQM

L'objet de ce chapitre est de présenter l'état de l'art de la mesure de la qualité d'une séquence vidéo. Cette revue se limitera aux méthodes pertinentes pour notre cas d'étude, c'est-à-dire le streaming vidéo mobile.

4.1 VQM basé sur le contenu

Dans [RIES2008], les auteurs proposent un modèle spécifique pour le streaming vidéo (H.264/AVC) mobile, basse résolution pour des connexions à faible bande passante. Ils se basent spécifiquement sur le contenu de la vidéo et des vecteurs de mouvement (MV).
La méthode divise la séquence vidéo en scènes qui sont alors classées suivant leur contenu (CC).Ces étapes se font au niveau du serveur de streaming. On transmet alors la vidéo avec les CC (Figure 5.a). Le client utilise alors la classe (CC) le bitrate et le framerate pour déterminer la qualité vidéo (Figure 5.b).

Les auteurs ont déterminé les contenus les plus utilisés lors du streaming vidéo en mobile, et ont défini 5 classes de contenu (avec des caractéristiques spécifiques) qui ont un impact différent sur la perception humaine :

- News (CC1) : région d'intérêts (ROI) +- 15% de la taille de l'écran. Parties en mouvement petites, arrière-plan statique,…
- Football (CC2) : contient de séquence grand angle avec un mouvement uniforme, arrière-plan coloré uniformément, objets (balle, joueurs) se déplacent rapidement.
- Cartoon (CC3): arrière-plan statique, objets se déplacent, les objets n'ont pas de caractère naturel
- Panorama(CC4) : séquences grand angle avec un mouvement global uniforme et dans une seule et même direction.
- Reste (CC5):changement de scène fréquent, beaucoup de mouvements localisés

La figure 5 montre le schéma de fonctionnement.

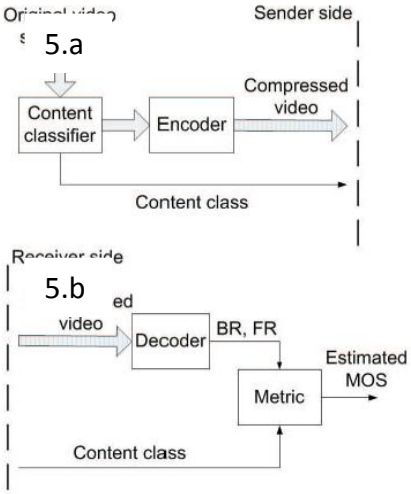


Figure 5 : Design de la méthode. [Ries2008]

La première étape consiste à diviser la séquence en différentes scènes. Les auteurs utilisent une méthode de détection de scènes basée sur un seuil dynamique [DIMOU2005] qu'ils ont adaptée pour améliorer la précision (97%).

L'étape suivante calcule les vecteurs de mouvement (MV). On divise l'image actuelle en blocs de 8x8 (*target blocks*). La différence relative en localisation entre le *target block* et le *matching block* (bloc qui correspond au target block de l'image précédente) est appelé le vecteur de mouvement (MV). Le processus de *block matching* correspond à comparer chaque bloc de l'image actuelle avec une certaine région de recherche dans l'image précédente (sur la composante luminance). Comme critère de correspondance, les auteurs calculent une somme de différences qui sont absolues (SAD). (L'algorithme n'est pas complètement détaillé à ce niveau)

En utilisant les MV de la scène, on calcule les paramètres de mouvement et de couleur pour la scène. On a les paramètres suivants :

- *Zero MV ratio* Nz : Pourcentage de MV's à zéro, détecte la proportion de région sans déplacement.
- *Mean MV size* n : proportion de la taille moyenne des non-zero MV's dans une trame normalisée sur la largeur de l'écran, en pourcentage. Donne la grandeur du mouvement global.
- *Horizontalness of movement* h : L'horizontalness est défini comme le pourcentage des MV's pointant dans une direction.
- *Uniformity of movement* d : Pourcentage des MV's pointant dans la direction dominante.
- *Greeness* g : pourcentage de pixels verts dans l'image.

On utilise ces paramètres dans une méthode statistique pour déterminer la classe à laquelle la scène en question appartient. Chaque classe de contenu dispose des caractéristiques

statistiques uniques pour les différents paramètres de mouvement et de couleur. On détermine des fonctions statistiques ECDF (*Empirical cumulative distribution function*) pour chaque paramètre à partir d'un ensemble typique de séquences vidéo pour chaque classe de contenu (Figure 6). On détermine également le modèle ECDF pour la séquence à classer.
On calcule alors la déviation maximale ($D_{cc\ max}$) d'une classe de contenu en considérant tous les paramètres.
Si Fn(x) est le modèle ECDF et F(x) est le ECDF de la séquence à classer, la différence maximale Dn entre $F_n(x)$ et F(x) :

$D_n = \max||F_n(x) - F(x)||$.

On compare maintenant chaque ECDF de la séquence à classer avec les ECDF des quatre classes de contenu. Si le D_n obtenu pour la classe testée est inférieur au $D_{cc\ max}$ pour chaque paramètre, alors la séquence appartient à cette classe.

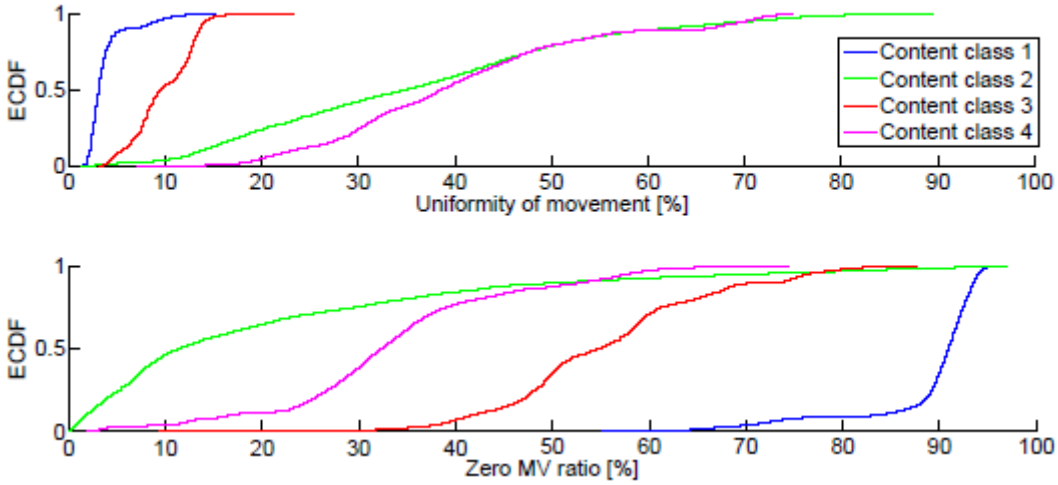


Figure 6: Modèle ECDF pour Nz et d pour un ensemble de séquences typiques [RIES2008].

Pour estimer la qualité au niveau du client, ils utilisent les informations de CC, du *bitrate* (BR) et du *framerate* (FR).

$$\widehat{MOS_{CC}} = A + B * BR + \frac{C}{BR} + D * FR + \frac{E}{FR}.$$

Avec les coefficients suivants (qui changent en fonction de la classe déterminée)

Coeff.	CC 1	CC 2	CC 3	CC 4	CC 5
A	4.0317	1.3033	4.3118	1.8094	1.0292
B	0	0.0157	0	0.0337	0.0290
C	−44.9873	0	−31.7755	0	0
D	0	0.0828	0.0604	0.0044	0
E	−0.5752	0	0	0	−1.6115

Tableau 7: Coefficients en fonction de la CC [RIES2008].

Leurs tests ont donné une performance de prédiction de métrique de 0.8303 par rapport au MOS (Pearson corrélation).

Evaluation du modèle :

- Le modèle a été développé pour le streaming vidéo mobile et correspond donc bien à notre cas. Ils prennent en compte les scénarios typiques pour le streaming vidéo mobile en considèrent les résolutions, bandes passantes typiques.
- Au niveau du client, la puissance de calcul nécessaire pour calculer la qualité vidéo est très petite. Et la métrique fournie a une bonne correspondance par rapport au MOS.
- Les auteurs parlent d'un modèle Reference Free, qui ne doit pas disposer de la vidéo originale. On peut remarquer qu'il s'agit plus tôt d'un ***Modèle reduced reference*** **(RR-VQM)** puisqu'on calcule une caractéristique (les CC) au niveau du serveur et on l'envoie en plus du vidéo au client.
- Comme on utilise des paramètres du flux vidéo (BR et FR) et des informations venant de la vidéo décodée (MV) il s'agit ici d'un modèle hybride.
- Les tests ont été réalisés avec des FR et BR différents, mais n'ont pas considéré les cas de perte de paquet, erreurs de bits, …

4.2 VQM basé sur le mouvement

Dans [RIES2007] et [RIES2008], les auteurs proposent un modèle adapté NR-VQM du modèle VQM basé sur le contenu. Ce modèle est spécifique pour le streaming vidéo (H.264/AVC) mobile, basse résolution pour des connexions à bande passante basse. Ils se basent spécifiquement sur les vecteurs de mouvement (MV) de la vidéo.
La méthode divise la séquence vidéo en scènes. On calcule alors les MV des images et les paramètres caractéristiques de mouvement de la scène (Z, S, N, U, BR). A partir des ces paramètres on détermine l'estimation de la qualité vidéo.

Comme décrit à la section précédente, la séquence est d'abord divisée en scènes, puis les MV sont calculés par la méthode SAD.

En utilisant les MV de la scène, on calcule les paramètres de mouvement. On a les paramètres suivants :

- *Zero MV ratio within one shot* Z : Z est égale à la moyenne du zero MV ratio Nz sur une séquence (shot). Voir section 4.1.
- *Mean MV size within one shot* V : V est égale à la moyenne du mean MV size sur une séquence (*shot*). Voir section 4.1.

- *Ratio of MV deviation within one shot* S : Proportion de la déviation standard MV par rapport à V, en pourcent.
- *Uniformity of movement* U : exprime la proportion des MV pointant dans la direction dominante sur une séquence (*shot*).
- *Average* BR : la BR est calculé comme la moyenne sur tout le stream.

On utilise ces paramètres pour déterminer la qualité vidéo objective ($\widehat{MOS_{MV}}$) :
$\widehat{MOS_{MV}} = a + b * BR + c * Z + d * S^e + f * V^2 + g * \ln(U) + h * S * V.$

Les auteurs ont déterminé les coefficients par régression, voir Tableau 8.

Coeff.	Value
a	4.631
b	8.966×10^{-3}
c	8.900×10^{-3}
d	-5.914×10^{-2}
e	0.783
f	-0.455
g	-5.272×10^{-2}
h	8.441×10^{-3}

Tableau 8 : coefficients pour déterminer la qualité vidéo [RIES2007].

Leurs tests ont donné une performance de prédiction de métrique de 0.8190 par rapport au MOS. (Pearson corrélation)

Evaluation du modèle :

- Le modèle a été développé pour le streaming vidéo mobile et correspond donc bien à notre cas. Il prend en compte les scénarios typiques pour le streaming vidéo mobile en considèrent les résolutions et bandes passantes typiques.
- La puissance de calcul nécessaire pour calculer la qualité vidéo est haute. C'est la détermination de vecteurs de mouvements MV qui est complexe.
- La métrique fournit une bonne correspondance par rapport au MOS.
- Il s'agit bien ici d'un modèle NR-VQM.
- Comme on utilise des paramètres du flux vidéo (bitrate) et des informations venant de la vidéo décodée (MV) il s'agit ici d'un modèle hybride.
- Les tests ont été réalisés avec des FR et BR différents, mais n'ont pas considéré les cas de perte de paquet, erreurs de bits, …

4.3 VQM basé sur la rupture de la fluidité vidéo/netteté

Dans [Pastrana2005], [Pastrana2006] et [Pastrana2007], les auteurs proposent un modèle basé sur les dégradations de fluidité vidéo (discontinuités de mouvement apparentes) et sur les dégradations de netteté, utilisable pour les services multimédia et les faibles bandes passantes.

Le modèle se base sur la combinaison de deux métriques différentes :

- Une métrique de fluidité (MOS_t)
- Une métrique *Clearness-scharpness* (netteté) : MOS_s

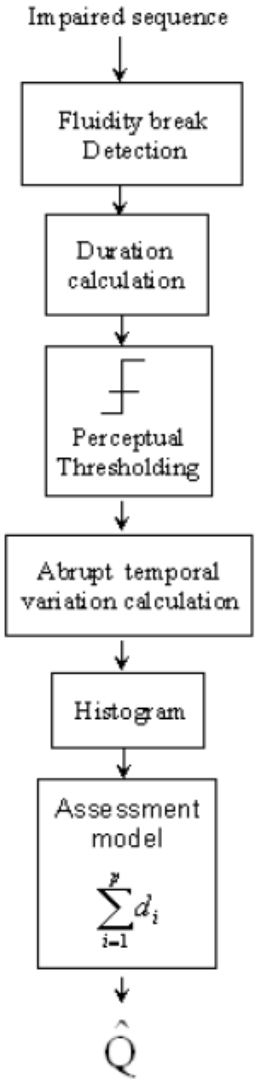
La métrique complète MOSv :

$$MOS_v = \frac{(v_{max} - v_{\min})}{1 + \left(\frac{b}{\gamma + (MOS_S * MOS_t)^c + \varepsilon}\right)^s} + v_{min}$$

Leurs tests ont donné une performance de prédiction de métrique de r=0.9 par rapport au MOS. (Pearson corrélation)

<u>La métrique de fluidité :</u>

Lorsqu'il y une perte de trame (*frame dropping*), cela est perçu à l'écran comme une image gelée (*frozen image*) suivie d'un déplacement abrupt d'objets. Le modèle prend en compte la durée et la densité temporelle des discontinuités (combien de fois sur une période d'analyse de 10 sec) ainsi qu'une fonction qui dépend de la densité de la dégradation pour calculer la métrique.



On détecte une discontinuité si la dérivée temporelle de la luminance de l'image est nulle :
Detec=1⇔Y(x,y,t) – Y(x,y,t+1) = 0.

Si la durée > seuil perceptuel (*threshold*) alors la discontinuité est perceptible

On calcule la variation de l'information temporelle à la fin de la discontinuité. (Dans [Pastrana2005] l'auteur propose un modèle plus simple qui ne tient pas compte de la variation de l'information temporelle).

$$\delta = f\left[\frac{RMSE(I_o, I_{o+1})}{val_{peak}}\right],$$

Après la fin de la détection, une distribution des durées des dégradations (histogramme) est calculée a postériori.

Finalement le modèle d'évaluation (*assessment*) produit la métrique de qualité vidéo pour la séquence.

Figure 7 : Schéma de la métrique de fluidité. [PASTRANA2007]

Le modèle d'évaluation (*Assessment model*) :
A partir de la durée et de la densité temporelle des discontinuités le modèle va calculer une métrique (MOSv) qui est en relation avec le MOS. (Pearson correlation : r=0.92)

mos_{ref} = MOS d'une séquence non dégradée
d_{total} : dégradation de la qualité totale limitée à dmax
$d_{pooling}$: dégradation sur une séquence de 10 sec.
t_g: temps de discontinuité
T_{min} : durée minimale d'une discontinuité perceptible (Tmin>threshold)
T_{max} : durée de la séquence : 10sec.
dt_g : contribution calculée de toutes les discontinuités qui ont une durée tg, $n(t_g)$ correspond à la distribution de la durée de discontinuité.
$\widetilde{q}(t_{g_i}, \delta_{t_{g_i}})$: est la fonction de qualité pour une discontinuité isolée de durée tg.
m_{max} et m_{min} : sont les valeurs de qualité extrêmes observées dans l'expérience.

$p(n(t_g))$: fonction exponentielle qui dépend de la distribution des durées. La valeur exponentielle peut varier de 1 à 2.

$$\hat{Q}=MOSv = mos_{ref} - d_{total},$$

$$d_{total} = min\{d_{pooling}, d_{max}\},$$

$$d_{pooling} = [\sum_{tg=Tmin}^{Tmax} d_{tg}]^{1/2},$$

$$d_{tg} = \sum_i \left[\hat{e}\left(t_{gi}, \delta_{t_{gi}}\right)\right]^{p(n(t_g))},$$

$$\hat{e}(t_{gi}, \delta_{t_{gi}}) = mos_{ref} - \hat{q}(t_{gi}, \delta_{t_{gi}}),$$

$$\hat{q}(t_{gi}, \delta_{t_{gi}}) = m_{max} - \frac{m_{max}-m_{min}}{1+\left(\frac{b}{t_g}\right)^s} - \delta_{t_{gi}},$$

$$p(n(t_g)) = p_{max} - \frac{p_{max}-p_{min}}{1+\left(\frac{c}{n(t_g)}\right)^r}.$$

La métrique de netteté :

Les auteurs se basent sur une propriété psycho visuelle selon laquelle le flou est en relation avec l'acuité des bords. Le flou peut être défini comme la sensation évoquée sur HVS (*Human Visual System*) par lissage de contours perceptibles. Dans [WATTMORGAN1983], les auteurs ont trouvé que la distance entre le max et le min. dans la seconde dérivée de la distribution de la lumière rétinale est en relation avec le flou perçu par HVS.
Sur cette base les auteurs [PASTRANA2007] proposent un indicateur sur la distance entre le max et le min de la dérivée seconde de la luminance des images. La métrique comprend également un seuil adaptable de gradient d'image, pour ne calculer que les gradients visibles.

On obtient les bords perceptibles en calculant les gradients d'image suivi par une application d'une fonction de seuil adaptable.

Calcul de la perte d'acuité : un filtrage de contour est appliqué aux images. On calcule alors le flou local sur des régions wx X wy (souvent 32X32 pixels). Les distances passant par 0 de la première dérivée de la luminance de l'image sont utilisées comme indicateur de la distance entre le max et le min dans la seconde dérivation de l'image rétinale qui est en relation avec la perception du flou.

On calcule alors le flou local sur un axe spatio-temporel (fenêtre de 160ms) wx X wy X wt.
L'indicateur final Bg est donné par le calcul de la moyenne.

Finalement une fonction de qualité calcule à partir de Bg la métrique MOSx. La Figure 8 montre le schéma du modèle.

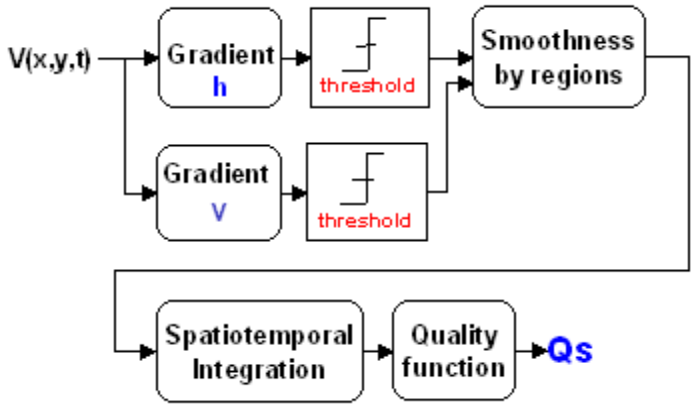


Figure 8 : Schéma du modèle de clearness/sharpness. [PASTRANA2007]

Evaluation du modèle :

- Le modèle prend en compte le streaming vidéo mobile et correspond donc bien à notre cas. Il prend en compte les scénarios typiques pour le streaming vidéo mobile en considérant les résolutions, bandes passantes typiques.
- La puissance de calcul nécessaire pour calculer la qualité vidéo est haute.
- La métrique fournit une bonne correspondance par rapport au MOS.
- Il s'agit bien ici d'un modèle NR-VQM.
- Comme on utilise seulement des informations venant de la vidéo décodée il s'agit d'une métrique de l'image.
- Les tests ont été réalisés avec des FR et BR différents. On a également fait des tests avec de pertes d'images (*frame skipping/dropping*)

4.4 VQM basé sur AMMF

Dans [LU2007], les auteurs proposent un modèle qui prend en compte les spécificités du streaming vidéo mobile, basse résolution pour des connexions à faible bande passante, perte d'images,…. Ils se basent sur une caractéristique temporelle de la vidéo qu'ils appellent AMMF (*Average Maximal Motion by Frame*) et du FR pour calculer la métrique.

L'AMMF peut être représenté comme une moyenne des valeurs d'offset de mouvement maximal, pour chaque image. Son représentation mathématique est la suivante :

$$AMMF = \sum_i \frac{\max(MotionMap_i)}{N-1}$$

Où $MotionMap_i$ correspond à l'ensemble des vecteurs de mouvement entre les images i et i+1. N est le nombre d'images connectées. Le map de vecteurs de mouvement peut être estimé par l'algorithme *optical flow* de [BLACK1996].

Sur base du AMMF les auteurs proposent une métrique de qualité vidéo :

$$MOS_p = 5 - (a_1 + a_2 * AMMF) * [\log(30) - \log(fr)]^{a3+a4*AMMF}$$

où fr : *frame rate*
Les coefficients a1-a4 ne sont pas donnés par les auteurs.
Leurs tests ont donné une performance de prédiction de métrique de r=0.9679 (CIF) et r=0.9752 (QCIF) par rapport au MOS. (Pearson corrélation)

Evaluation du modèle :

- Le modèle prend en compte le streaming vidéo mobile et correspond donc bien à notre cas. Ils prennent en compte les scénarios typiques pour le streaming vidéo mobile en considérant les résolutions, bandes passantes typiques.
- La puissance de calcul nécessaire pour calculer la qualité vidéo réside surtout dans l'algorithme d'optical flow.
- La métrique fournit une bonne correspondance par rapport au MOS. (On n'a pas introduit d'autres dégradations spatiales dans les séquences de test)
- Il s'agit bien ici d'un modèle NR-VQM.
- Comme on utilise des paramètres du flux vidéo (frame rate) et des informations venant de la vidéo décodée (AMMF) il s'agit ici d'un modèle hybride.
- Les tests ont été réalisés avec des FR et BR différents. On a également fait des tests avec de pertes d'images (frame *skipping/dropping*). On n'a pas introduit d'autres dégradations spatiales dans les séquences de test.

4.5 VQM basé sur Flou/Bloc/Jerkiness

Dans [LIU2008], les auteurs proposent un modèle basé sur plusieurs dégradations de la vidéo introduites à la section 3.3: le *Blurring* (flou), *Block Distorsion* (bloc), *Jerkiness/Jitter*. La

métrique finale est une combinaison hybride des métriques de ces dégradations qu'ils appellent WHRFVQM.

Le VQM *Blurring* (flou)

Les auteurs se basent sur le fait que les personnes sont plus sensibles aux bords d'une image vidéo et que typiquement une des caractéristiques du flou est la netteté réduite des bords.
Les auteurs utilisent l'algorithme suivant (dans le sens horizontal et vertical) :

1. Appliquer un filtre Sobel horizontal sur l'image vidéo, pour avoir une image des bords.
2. Fixer le seuil horizontal prédéfini (Thorizontal)
3. Si la valeur du Pixel de l'image filtrée est plus grande que Thorizontal, le pixel est mis à Hor_Edge_Sign.
4. Calculer la largeur (Hor_Blur_D) pour chaque Hor_edge_Sign le long de sa ligne horizontale. [MARZILANO2002]
5. Calculer la largeur moyenne Hor_Blur_$D_{average}$ pour tous les Hor_Edge_Sign

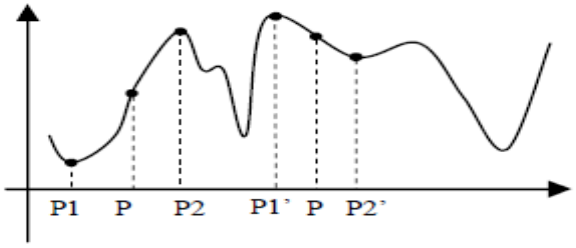


Figure 9 : Largeur pour chaque signature de bord [LIU2008].
Calcul du Hor_Blur_D pour le bord P, puis calcul de la moyenne de tous les Hor_Blur_D :

$$Hor\_Blur\_D = |P1 - P2|$$

$$Hor\_Blur\_D_{average} = (\sum_{n=0}^{N} Hor\_Blur\_D(n))/N$$

Le même algorithme est effectué pour la direction verticale et finalement la métrique est calculée avec :

$$VQM\_Blur = wb_h * Hor\_Blur\_D_{average} + wb_v * Ver\_Blur\_D_{average}$$

$$wb_h + wb_v = 1$$

Le VQM *Blocking*

Les auteurs partent du principe que l'effet de blocking arrive toujours à la frontière des blocs (en raison de la conception Block-based des CODEC MPEG et H.26x) et continuent vers l'intérieur du bloc car des valeurs de pixel ont été adaptées à la moyenne. Ils utilisent alors une détection *Boundary smoothness* (frontière lisse) entre des blocs 8 X 8 (a) et une détection de visibilité des blocs (b) pour calculer une métrique de *blocking* générale.

a.**Détection de *boundary smoothnes* entre blocs 8X8 :**
Normalement les différences entre les pixels voisins se situant des 2 côtés de la frontière entre 2 blocs seront petites, même s'il y a un changement de scène. Lorsque l'effet de *blocking* est visible, cette différence entre pixels sera plus grande.
Les auteurs calculent la *boundary smoothness* entre chaque paire de 8X8 blocs, et calculent alors la moyenne pour avoir une métrique de *boundary smooth* globale. Ceci sera fait suivant la direction horizontale et verticale.

$$Hor\_Boundary\_SMooth_{average} = \sum_{j=0}^{bn} \sum_{i=0}^{7} \frac{abs(x_i - y_i)}{bn}$$

Hor_Boundary_smoothaverage : moyenne des boundary smoothness
bn : horizontal boundary number
xi et yi sont des valeurs de pixels voisins pour 2 blocs différents

On peut alors calculer la valeur globale :
$Block\_Smooth = wbs_h * Ver\_Boundary\_Smooth_{average} +$
$wbs_v * Hor\_Boundary\_smooth_{average}$

$wbs_h + wbs_v = 1$

wbs_h : indice pondéré pour la direction horizontale
wbs_v : indice pondéré pour la direction horizontale

b.**Détection de visibilité des blocs :**
Suivant les auteurs, la visibilité d'un bloc est déterminée par le contraste entre le gradient local et le gradient moyen des pixels adjacents.

$D_{H,norm}$: *normalized horizontal gradient* : ratio entre le gradient absolu et le gradient moyen calculé sur N pixels adjacents vers la droite et la gauche
i : line
j : pixel position

$$D_{H,norm}(i,j) = \frac{|I(i+1,j) - I(i,j)|}{\frac{1}{2N}\sum_{n=N....N,n\neq 0}|I(i+n+1,j) - I(i+n,j)|}$$

Comme les bords des blocs arrivent à des intervalles réguliers dans la direction horizontale et verticale, on peut faire la somme de $D_{H,norm}$ sur toutes les lignes de l'image :

$S_H(i) = \sum_{j=1}^{nl} D_{H,norm}(i,j)$

La grandeur visuelle des artefacts de blocking peut être déterminée par :

$$BS = \frac{S_H(block)}{S_H(non-block)}$$

Finalement la métrique globale est calculé par :

$$VQM\_Blocking = w_{smooth} * Block\_Smooth + wb_s * BS$$
$$w_{smooth} + wb_s = 1$$

où w_{smooth} et w_{bs} sont des indices pondérés.

Le VQM *Jerkiness/jitter*

Pour les auteurs la perte de paquets vidéo résulte souvent dans une perte d'information de slice. Un slice est une région de l'image qui est encodée indépendamment des autres régions de l'image. Cette perte d'information résulte à son tour en une corruption de l'information visuelle d'un macrobloc (MB) et du bord du slice. Généralement les décodeurs remplacent les paquets manquants en utilisant le MB de l'image précédente. Donc on aura une discontinuité visible s'il y a beaucoup de mouvement entre les images consécutives.

L'idée des auteurs est de calculer l'*energy strength* le long de chaque bord de slice :

$$s_{temp}(j) = \left|P(i-1) - P(i+1)\right) * F|$$
$$s'_{temp}(j) = \left|P(i-2) - P(i)\right) * F|$$

p(i) : i-eme ligne de l'image
i : dernière ligne de chaque MB (16 ,32 ,…m-16)
j : est le numéro du bord qui est égal à i/16.
S_{temp} : *boundary strength* pour le MB j et j+1.
S'_{temp} : *boundary strength* voisin de S_{temp}.
* : opération de convolution

Pour éviter le *boundary noise*, ils définissent un *pre-threshold* (T)

S(k) = { *1: if $S_{temp}(k) > T, K = 1,2 \ldots\ldots m$* *}*
0: else

S'(k) ={ *1: if $S'_{temp}(k) > T, k = 1,2\ldots\ldots m$* *}*
0: else

S(k) et S'(k) doivent avoir des frontières similaires dans une image sans dégradations.
L'artefact dû à la perte des paquets le long de la ligne du MB j est calculé comme :

$$E(j) = \sum_i S(j)(i) - S'(j)(i)$$

Et la métrique de jerkiness =

$$F = \sum_j E^2(j)$$

Finalement les auteurs définissent la métrique globale :

$$WHRFVQM = \alpha * X_{blur} + \beta * Y_{block} + \gamma * Z_{jerk} + \lambda$$

Leurs tests ont donné une performance de prédiction de métrique de r=0.9002 par rapport au MOS. Les séquences utilisées étaient au format de télévision standard (SDTV).

Evaluation du modèle :

- Le modèle prend en compte les dégradations majeures qui ont un effet sur le MOS (Block distorsion, blurring, jerkiness). Il n'a pas été spécialement conçu pour le streaming multimédia
- La puissance de calcul nécessaire pour calculer la qualité vidéo découle de la puissance de calcul nécessaire de calcul de chaque VQM, qui peut être grande.
- La métrique fournit une bonne correspondance par rapport au MOS. (Ils ont utilisé des séquences de résolution TV standard)
- Il s'agit bien ici d'un modèle NR-VQM.
- Comme on utilise seulement des informations venant de la vidéo décodée il s'agit d'une métrique de l'image.

4.6 VQM basé sur l'extraction de paramètres par le décodeur

Dans [ROSSHOLM2008], les auteurs proposent un modèle spécifique pour le streaming vidéo (H.264/AVC) mobile, basse résolution pour des connexions à faible bande passante. Le modèle est basé sur l'utilisation des paramètres du bitstream vidéo.

Le modèle mathématique est le suivant :
Suivant les auteurs, on a une matrice d'observation X=[X1 X2 …Xn], où X1..Xn sont un nombre de feature vectors. Chaque feature vector consiste en des paramètres video extraits du bitstream noté x1-xk.
Les mesures de qualité correspondantes, PSNR, PEVQ, SSIM, VSSIM et NTIM (ce sont des VQM) correspondent alors à Y= [Y1 Y2 … YN]. X et Y peuvent être considérés comme données d'entraînement pour un problème de classification ou régression. On veut trouver une fonction Z = f(X) qui lie les valeurs données dans X vers une valeur Z, qui est une estimation d'un VQM, par exemple le PSNR.
Les auteurs choisissent la technique de régression multi-linéaire pour résoudre le problème.
Le modèle multi-linéaire est formulé comme :

$Y = \beta X + \varepsilon$
où ε représente la variation non prédite. La régression multi-linéaire estime les valeurs pour β noté $\hat{\beta}$ qui peuvent être utilisées pour prédire Z comme

$$\hat{Z} = \widehat{\beta_0} + \widehat{\beta_1}\, x_1 + \widehat{\beta_2}\, x_2 + \cdots + \widehat{\beta_k}\, x_k$$

On utilise le coefficient de corrélation linéaire de Pearson pour évaluer la métrique prédite :

$$rp = \frac{\sum(\hat{Z}_\iota - \hat{Z}_{mean})(Z_i - Z_{mean})}{\sqrt{\sum(\hat{Z}_\iota - \hat{Z}_{mean})^2}\ \sqrt{\sum(Z_i - Z_{mean})^2}}$$

Les auteurs ont déterminé les paramètres en mesurant leur influence sur les métriques qui sont représentées à la Figure 10 :

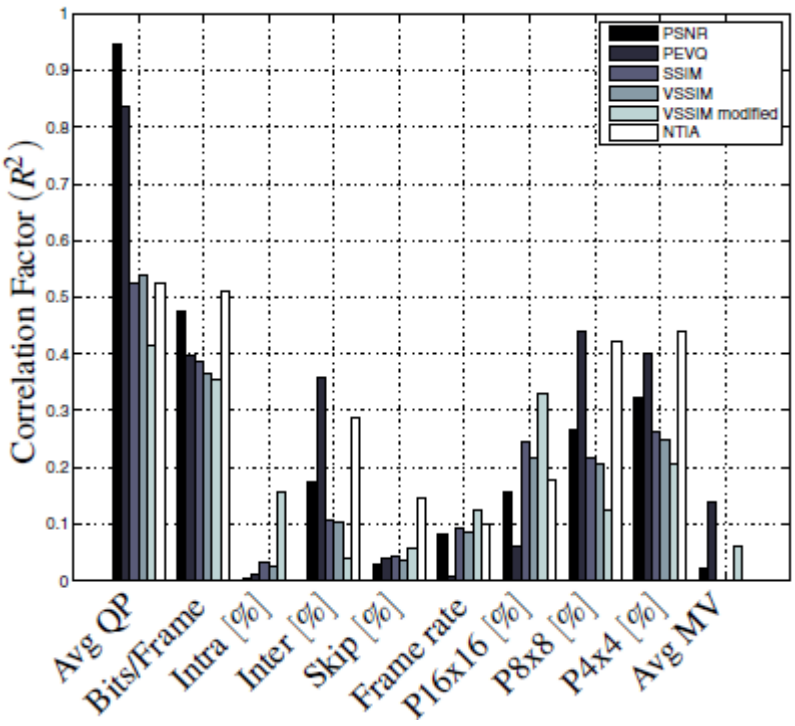


Figure 10 : Le facteur de corrélation R2 entre chaque paramètre et les métriques utilisées [ROSSHOLM2008].

Les paramètres retenus et leurs correspondances :

$\hat{\beta}_k$	Parameter
$\hat{\beta}_0$	Constant
$\hat{\beta}_1$	Avg QP
$\hat{\beta}_2$	Bits/Frame
$\hat{\beta}_3$	Frame rate
$\hat{\beta}_4$	P16x16 [%]
$\hat{\beta}_5$	P8x8 [%]
$\hat{\beta}_6$	P4x4 [%]
$\hat{\beta}_7$	Avg MV

Tableau 9 : Mapping des Beta aux paramètres utilisés dans la régression [ROSSHOLM2008].

P16x16 [%]: *Number of inter blocs of size* 16x16, les inter blocs étant des blocs contenu dans des trames P ou B
P8x8 [%]: *Number of inter blocs of size* 8x8, 16x8 et 8x16
P4x4 [%]: *Number of inter blocs of size* 4x4, 8x4 and 4x8
Avg MV: *Average motion vector length*

Metrics	$\hat{\beta}_0$	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$	$\hat{\beta}_4$	$\hat{\beta}_5$	$\hat{\beta}_6$	$\hat{\beta}_7$	Scale
PSNR	66.89	-0.92	-0.07	-0.03	-0.01	-0.09	-0.07	0.01	$1.0\exp{-0}$
SSIM	109.67	-0.46	0.16	-0.03	-0.38	0.26	0.49	-0.03	$1.0\exp{-2}$
VSSIM	112.76	-0.52	0.10	-0.01	-0.33	0.19	0.41	-0.01	$1.0\exp{-2}$
VSSIM modified	111.74	-0.48	0.00	0.00	-0.33	0.17	0.32	0.37	$1.0\exp{-2}$
NTIA	66.13	-0.66	1.23	0.39	-0.82	1.34	-1.61	0.41	$1.0\exp{-2}$
PEVQ	55.94	-0.92	0.14	-0.21	-0.07	0.23	0.26	-0.26	$1.0\exp{-1}$

Tableau 10 : Les valeurs des $Beta_i$ pour les différentes métriques [ROSSHOLM2008].

Metric	r_P
PSNR	0.99
SSIM	0.62
VSSIM	0.61
VSSIM modified	0.71
NTIA	0.74
PEVQ	0.95

Tableau 11 : Le coefficient de corrélation pearson, rp, pour la prédiction des métriques [ROSSHOLM2008].

Le Tableau 11 montre que le modèle a une bonne corrélation avec les VQM PSNR et PEVQ.
Il faut noter que le VQM PEVQ est un VQM *Full Reference* et fait partie de la recommandation ITU-T J.247 (*Objective perceptual multimedia video quality measurement in the presence of a full reference*).

Evaluation du modèle :

- Le modèle prend en compte le streaming vidéo mobile et correspond donc bien à notre cas. Il prend en compte les scénarios typiques pour le streaming vidéo mobile en considèrent les résolutions, bandes passantes typiques.
- La puissance de calcul nécessaire pour calculer la qualité vidéo est petite.
- La métrique fournit une bonne correspondance par rapport à la VQM PEVQ, dans les conditions testées(H.264).
- Il s'agit bien ici d'un modèle NR-VQM.
- Comme on utilise seulement des paramètres du flux vidéo (*bitstream*) il s'agit d'une métrique basée sur le paquet- et *bitstream*.
- Les tests ont été réalisés avec des FR et BR différents.
- Pour des CODEC différents des nouvelles valeurs de Beta doivent être calculées.
- La métrique dépend du CODEC.

4.7 VQM basé sur le Deep Packet Inspection

Dans [Winkler2008], les auteurs proposent un modèle basé sur l'utilisation des paramètres du *bitstream* vidéo. Ils présentent leur VQM : *V-Factor* qui utilise le *Deep Packet Inspection*. (Inspection de paquets en profondeur). Cette métrique est principalement orientée vers le streaming vidéo MPEG-2 et H.264 sur des réseaux IP.

Suivant les auteurs, leur modèle analyse le *bitstream* en temps réel pour collecter des paramètres statiques (taille de l'image, FR) et dynamiques (variation de la quantification,…)
La prédiction de la qualité vidéo est basée sur :

- L'impact dû à des caractéristiques du contenu, les mécanismes de compression et des contraintes de bande passante sur la dégradation de la vidéo
- L'impact du *jitter*, *delay* et perte de paquets sur la vidéo.

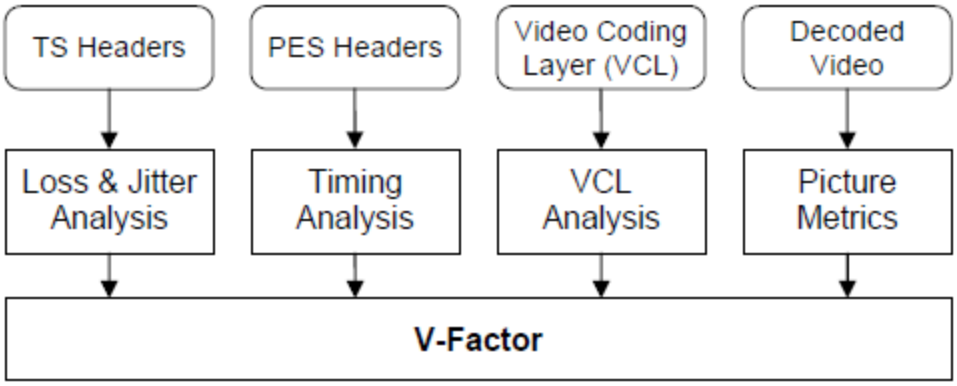


Figure 11 : L'analyse du flux vidéo sur plusieurs niveaux [WINKLER2008].

La figure 11 montre que le V-Factor analyse différentes sections du flux vidéo. Le TS (*Transport Stream*) est le niveau le plus bas et comporte des informations de multiplexage, de synchronisation et de correction d'erreur. Le PES (*Packetized Elementary Stream*) est le niveau supérieur et comporte un flux élémentaire (une vidéo encodée par exemple). Le niveau supérieur est le VCL (*Video Coding Layer*), qui sert à représenter efficacement le contenu vidéo.

Le modèle pour la mesure objective des dégradations vidéo se base sur le modèle de [VERSCHEURE1999], traitant l'impact de la PLR (*Packet Loss Rate*), MPEG-2 *quantizer scale* et MPQM (*Moving Picture Quality Metric*). Les auteurs ont généralisé ces modèles à H.264 et apporté des améliorations.

Analyse Bitstream :
La figure 12 montre le modèle de complexité H.264 VCL, qui réalise des analyses au niveau du VCL.

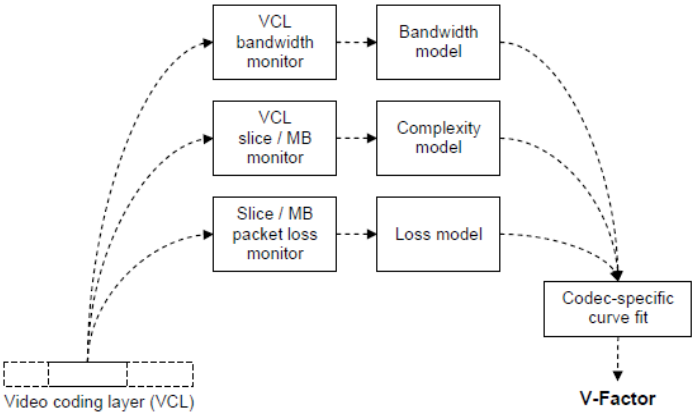


Figure 12 : Modèle de complexité H.264 VCL [WINKLER2008].

L'inspection des paquets permet également d'établir des modèles de la bande passante pour un stream vidéo et d'un modèle de jitter.

Les auteurs ne décrivent pas leur modèle en détail.

Evaluation du modèle :

- Le modèle prend en compte le streaming vidéo.
- La puissance CPU nécessaire pour calculer la qualité vidéo est faible.
- Les auteurs ne donnent pas de correspondance par rapport au MOS.
- Il s'agit bien ici d'un modèle NR-VQM.
- Comme on utilise des paramètres du flux vidéo (frame rate) et des informations venant de la vidéo décodée il s'agit ici d'un modèle hybride.
- Pour des CODEC différents, la métrique doit être adaptée.

4.8 Comparaison et choix de modèles

\VQM	Contenu [RIES2008]	Mouvement [RIES2007, 2008]	Fluidité/ netteté [PASTRANA 2006, 2007]	AMMF [LU2007]	Flou/bloc /jerkiness [LIU2008]	Paramètres bitstream [ROSSHOLM2008]	Deep Packet Inspection [Winkler2008]
Type VQM	RR-VQM	NR-VQM	NR-VQM	NR-VQM	RR-VQM	NR-VQM	NR-VQM
Métrique	Hybride	Hybride	Image	Hybride	Image	Packet - Bitstream	Packet- Bitstream /Hybride
Puissance de calcul nécessaire	basse	Haute	Haute	Haute	Haute	Basse	Basse
Corrélation* MOS	0,8303	0,8190	0,9	> 0,9679	0,9002	0,95 par rapport au VQM PEVQ	?
Concept	Motion vectors et Content classes	Motion vectors	VQM Fluidité : frame dropping + VQM netteté	AMMF : Motion Map	VQM flou + bloc + jerkiness	Extraction param. Bistream /régression coefficients	Extraction en profondeur / plusieurs modèles de prédiction
Paramètres Bitstream	Bitrate, Frame rate	Bitrate	Non	Frame rate	Non	Avg QP, Avg MV, Bits/Frame, Frame rate	Multiples
Low-Bitrate	Testé	Testé	Testé	Testé	Non Testé	Testé	?
Low-Resolution	Testé	Testé	Testé	Testé	Non Testé	Testé	?
Paquet drop	Non testé	Non testé	Testé	Testé	Non testé	Non testé	?
Artefacts Blocking	Non calculé	Non calculé	Non Calculé	Non calculé	Calculé	Non calculé	Non Calculé
Artefacts Blurring	Non calculé	Non calculé	Calculé	Non calculé	Calculé	Nn calculé	Non Calculé
Jerkiness/ jitter	Non calculé	Non calculé	Calculé	Non calculé	Calculé	Non calculé	Calcul indirect
CODEC	Indépendant	Indépendant	Indépendant	Indépendant	Indépendant	Dépendant	Dépendant

Tableau 12: Comparaison des modèles VQM.

*: les auteurs ont généralement utilisé des séquences vidéo contenant seulement les dégradations gérées par le VQM en question.

Le Tableau 12 montre une comparaison entre les différents modèles VQM.

Choix des modèles :
Pour pouvoir choisir les modèles les plus prometteurs, il faut tenir compte de l'utilisation prévue. Dans les chapitres 2 et 3 on a déterminé les spécificités du streaming vidéo mobile. Les points les plus importants étaient :

- Bande passante limitée (peut descendre en dessous de 180 kbps)
- Résolution de la vidéo petite (QCIF, CIF, SIF,…)
- Connexion radio (perte de paquets, erreur de bits)
- Puissance CPU des terminaux mobiles limités

On peut également noter que la fonction que le VQM devra effectuer est une fonction de monitoring de la qualité vidéo (Il ne s'agit pas d'évaluer la qualité d'encodage de la vidéo). Il s'agit donc surtout de problèmes de diminution de la bande passante au niveau du client ou des perturbations de la liaison radio, qui donnent lieu à des pertes de paquets IP. Ce qui peut engendrer le phénomène de *jerkiness/jitter*.

Ceci nous amène aux critères de sélection pour les modèles VQM pour notre cas d'étude:

-modèle disposant d'une bonne corrélation par rapport au MOS.
-modèle tenant compte d'une bande passante limitée
-modèle adapté aux résolutions vidéo mobiles
-modèle tenant compte du « *jerkiness/jitter* »
-modèle nécessitant une puissance de calcul limitée

-Bien que le VQM contenu (Section 4.1) puisse être intéressant au point de vue puissance de calcul nécessaire, il s'agit ici d'un RR-VQM, qui nécessite la transmission d'informations complémentaires. Donc ce VQM est moins adapté à notre cas.

-Le VQM Mouvement (Section 4.2) donne des résultats intéressants, mais il ne tient pas compte du « *jerkiness/jitter* »

-Le VQM fluidité/netteté (Section 4.3) donne des résultats intéressants, il tient en autre compte du *jerkiness*. On peut également dissocier l'algorithme de fluidité de l'algorithme de netteté, pour réduire la puissance de calcul nécessaire.

-Le VQM AMMF (Section 4.4) donne également des résultats intéressants, mais le « *jerkiness/jitter* » n'est pas pris en compte.

-La métrique la plus complète est certainement le VQM de Flou/bloc/*jerkiness* (Section 4.5) qui évalue les dégradations majeures qui ont un impact sur la qualité vidéo subjective. L'inconvénient est la puissance de calcul nécessaire. Il n'a pas été testé sur de la vidéo basse resolution/bitrate.

-Le VQM paramètres bitstream (Section 4.7) dispose d'une puissance de calcul nécessaire basse. Il a un concept intéressant et a une bonne corrélation avec le VQM PEVQ. Il est par contre dépendant du CODEC utilisé.

-Le *Deep Packet Inspection* (Section 4.8) pourrait certainement être une réelle alternative, (extraction des paramètres en profondeur/ plusieurs modèles de prédiction, puissance de calcul nécessaire basse,…) mais l'information disponible sur ce VQM est limitée. Je pense que ce type de VQM a de bonnes perspectives pour le futur.

Le VQM fluidité/netteté et le VQM Flou/bloc/*jerkiness* sont les seules à calculer le *jerkiness/jitter*, une des dégradations majeures pour le streaming vidéo. Le VQM Flou/bloc/*jerkiness* n'ayant pas été testé sur de la vidéo LOW résolution/bitrate le VQM fluidité/netteté correspond le mieux aux critères de sélection établis.
Pour améliorer d'avantage la correspondance au critère de la puissance de calcul limitée, on choisit de dissocier le VQM fluidité/netteté et de n'utiliser que la métrique fluidité du VQM fluidité/netteté.

On choisit donc le VQM basé sur la rupture de la fluidité vidéo (Section 4.3).

5 Analyse du feed-back MOS vidéo via RTCP/RTSP

Dans les chapitres précédents, on a déterminé une méthode pour pouvoir calculer le MOS Vidéo. Ce chapitre va passer en revue les possibilités de renvoyer cette métrique MOS vidéo vers le serveur de streaming via les protocoles RTCP/RTSP.

Pour chacun de ces protocoles, on va expliquer l'utilisation du protocole, déterminer s'il existe déjà une possibilité standardisée pour renvoyer le MOS vidéo, et déterminer s'il y a des propositions non standardisées pour le feed-back du MOS Vidéo.
Finalement, en tenant compte du contexte du mémoire, on va choisir une des possibilités pour le feed-back du MOS vidéo.

5.1 Le protocole RTCP

5.1.1 Généralités

Le standard RFC 3550 [RFC3550] spécifie le protocole RTP Control Protocol (RTCP). RTCP est un protocole de contrôle associé à RTP. Ce protocole réalise les fonctions suivantes :

-fournir un feed-back sur la qualité de la distribution des données (pour le contrôle du flux RTP et de la congestion)

-transmettre un identificateur pour une source RTP appelé le CNAME.

-déterminer la fréquence (variable) d'envoi des paquets de contrôle que chaque participant doit envoyer régulièrement. Le nombre de participants et le débit media RTP sont utilisés pour déterminer cette fréquence d'envoi.

Plusieurs types de paquets ont été définis par le RFC 3550 pour pouvoir transporter une information de contrôle variée :

-SR : *Sender Report*, statistiques de transmission et de réception transmises par des expéditeurs actifs (*active senders*). Le type de paquet (PT) a la valeur 200.

-RR : *Receiver Report*, statistiques de réception transmises par des participants qui ne sont pas des expéditeurs actifs. Le PT a la valeur 201.

-SDES : éléments de description des sources avec le CNAME. Le PT a la valeur 202.

-BYE : indique la fin d'une participation. Le PT a la valeur 203.

-APP : fonction spécifique à des applications (*application specific*) qui ne sont pas standardisées. Le PT a la valeur 204.

Le standard RTCP permet d'envoyer plusieurs paquets RTCP comme paquet RTCP composé (*compound*) dans un paquet unique du protocole sous-jacent.

Dans notre contexte, le *Receiver Report* (RR) qui est envoyé régulièrement par le client vers le serveur de streaming est le seul type de paquet RTCP standardisé qui renvoie des statistiques.
La Figure 13 montre sa structure.

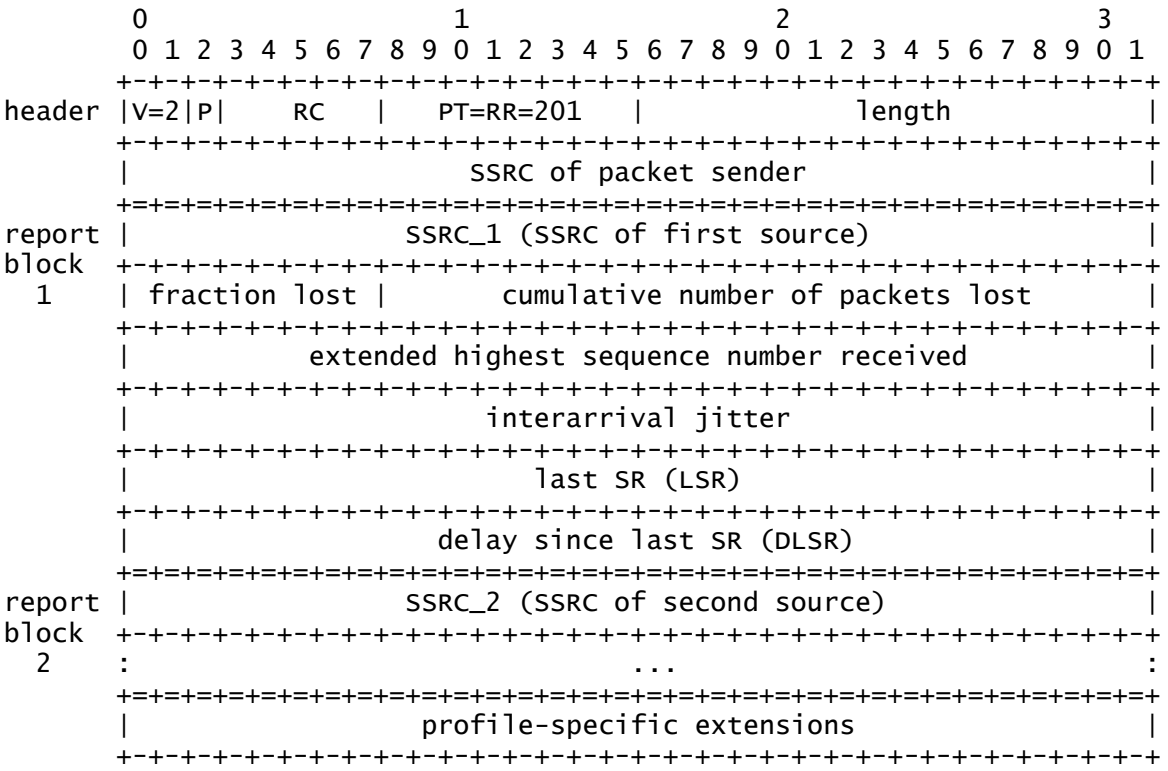


Figure 13 : structure d'un *Receiver Report* [RFC3550].

Le *Receiver Report* consiste en 2 sections avec une section supplémentaire possible (extensions spécifiques au profil) :

-le *header*, en-tête
-zéro ou plusieurs blocs de rapport
-un bloc d'extension spécifique au profil

Voici la signification des champs principaux.
Pour l'en-tête :
V : identifie la version de RTP/RTCP, ici 2.
P : *Padding*, si P=1, le paquet RTCP contient des octets de padding, qui ne font pas partie de l'information de contrôle.
RC : nombre de blocs de rapport de réception (reception report block) contenu dans ce paquet.
PT : valeur du type de paquet, 201 pour Receiver Report.
Length : longueur de ce paquet RTCP en mots de 32-bits - 1.
SSRC : identificateur de la source de synchronisation de l'expéditeur du paquet.

Pour le bloc de rapport :
SSRC_n : identificateur de la source pour laquelle l'information du bloc de rapport est destinée.
Fraction lost : la fraction de paquets RTP envoyés par SSRC_n perdus depuis le dernier paquet SR ou RR.
Cumulative number of packets lost: le nombre total de paquets RTP de SSRC_n perdus.
Interarrival jitter : estimation de la variance statistique de l'intervalle du temps de réception entre les paquets RTP.
Delay since last SR : délai entre le dernier paquet SR de la source SSRC_n et l'envoi du bloc de report de réception présent.

Ces champs permettent au serveur de calculer des statistiques et de déterminer des problèmes de flux ou de congestion du réseau. Mais le Receiver Report ne dispose pas d'un champ spécifique pour renvoyer la métrique MOS Vidéo.

On peut également noter que l'intervalle de temps entre deux paquets RTCP varie. Le standard [RFC3550] défini un algorithme qui utilise le nombre de participants, la bande passante et un nombre aléatoire entre 0,5 et 1,5 pour recalculer à chaque fois un nouveau intervalle.

5.1.2 Paquet RTCP "*Application-Defined*"

Une possibilité pour envoyer le MOS Vidéo est d'utiliser des types de paquets RTCP *Application-Defined* (PT=204). Le standard rfc3550 [RFC3550], définit un type de paquet RTCP "*Application-defined*", qui dispose d'un champ data que l'on peut structurer librement.
Ce type de paquet est normalement prévu pour des utilisations expérimentales lors du développement de nouvelles applications sans qu'il ne soit nécessaire d'enregistrer une nouvelle valeur de type de paquet (PT).

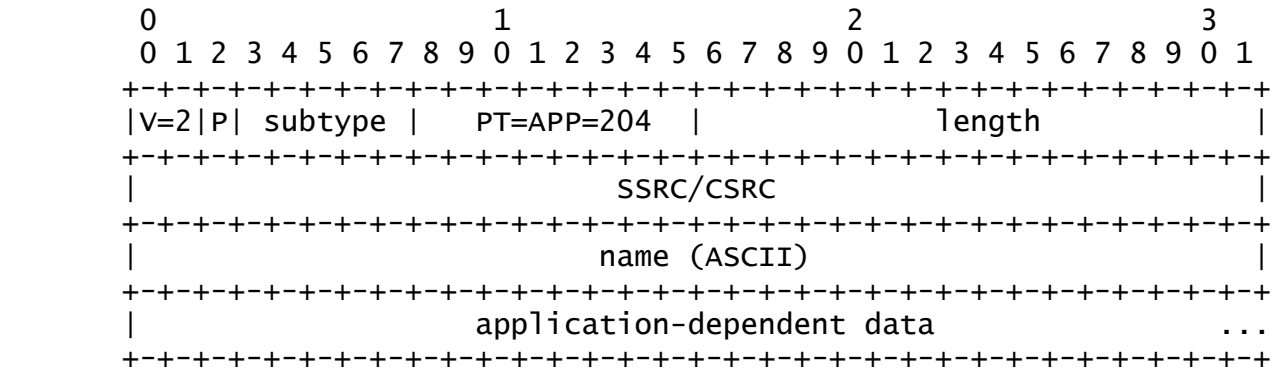


Figure 14 : Structure d'un paquet RTCP *APP-Defined* [RFC3550]

La Figure 14 montre la structure d'un paquet RCP *Application Defined*. La version (v), le *padding* (P), la longueur et le SSRC correspondent aux champs d'un paquet RTCP RR. Le type de paquet (PT) est 204. Le name correspond au nom donné à des paquets Application defined.
Le champ *data* correspond à une structure de données qu'on peut définir.

Pour le feed-back du MOS Vidéo en unicast on peut définir le paquet de la manière suivante :

-name= 'MOSV'

-data=valeur du MOS Vidéo en 16 bits

Il est clair que la structure du champ data pourra encore être affinée pour contenir d'autres données (MOS audio,…).
On peut alors envoyer ce paquet avec un paquet RTCP RR en paquet composé.

5.1.3 Le protocole RTCP XR

Dans le RFC 3611 [RFC3611], les auteurs ont défini un nouveau type de paquet RTCP, le rapport étendu (*Extended report*, XR) RTCP XR. Les paquets XR permettent de transmettre des informations qui vont plus loin que les informations déjà contenues dans les blocs de rapports de réception des RR et SR.

On peut distinguer trois catégories de type de bloc de rapport :

-rapports sur les paquets RTP reçus ou perdus. Il s'agit de rapports concernant la réception et la perte de paquets RTP, la réception des paquets dupliqués, une liste d'estampilles (*timestamps*) de réception de paquets RTP.

-rapports qui transmettent des informations de temps de référence entre participants. Il s'agit de rapports concernant les timestamps du côté récepteur, et de délais de réception qui permettent entre autres de calculer le temps aller-retour (*round-trip time*).

-rapports qui transmettent des métriques sur la réception de paquets. Il s'agit de rapports concernant des statistiques sur les numéros de séquence de paquets RTP, sur la perte, le *jitter*, le TTL. On a également des rapports qui contiennent des métriques pour le monitoring des appels Voice over IP (VOIP).

La Figure 15 montre la structure générale d'un paquet RTCP XR. La valeur du type de paquet RTCP est le 207 (PT). La majorité des autres champs correspondent aux champs d'un paquet RTCP. Le report bloc contient un ou plusieurs blocs de rapport RTCP XR.

Comme pour les paquets RTCP normaux, les paquets RTCP XR peuvent être regroupés en un paquet composé.

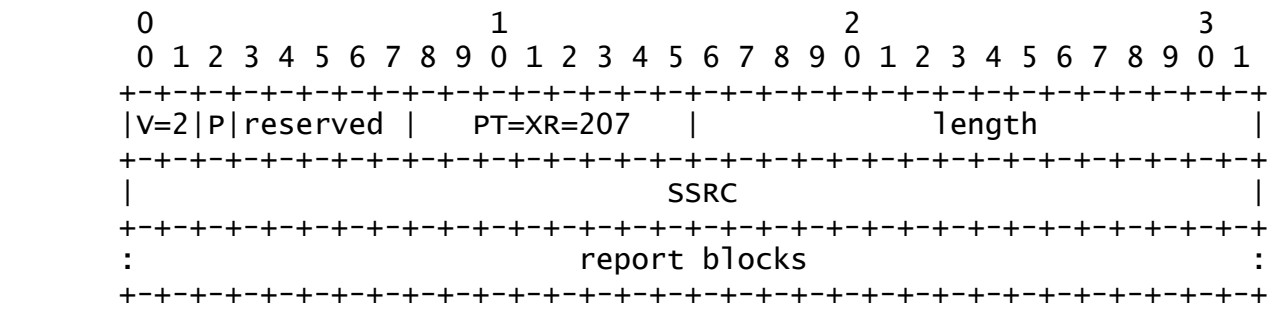


Figure 15 : la structure générale d'un paquet RTCP XR [RFC3611].

La structure d'un bloc de rapport étendu est reprise dans la Figure 16. On retrouve les champs suivants :

BT : type de bloc, identifie le format de bloc. Sept formats de bloc sont définis suivant le RFC 3611, d'autres pourraient être définis.

Type specific : le contenu de ce champ est déterminé par la définition du type de bloc choisi.

Type-specific bloc contents : le contenu de ce champ est déterminé par la définition du type de bloc choisi.

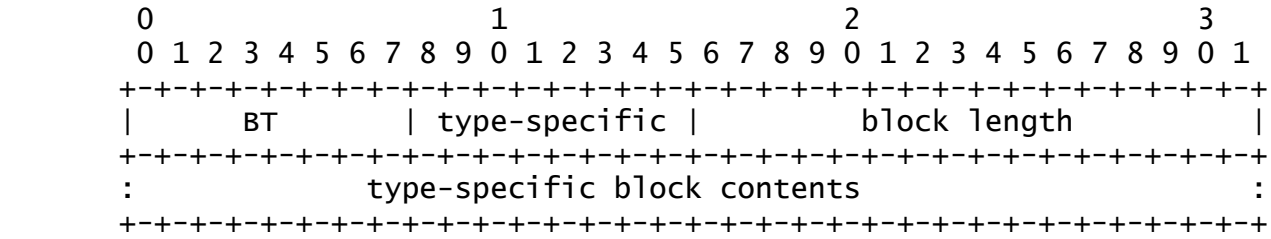


Figure 16 : la structure générale d'un bloc de rapport étendu [RFC3611].

Dans notre contexte, aucun des sept blocs de rapport étendu défini par le RFC 3611 ne permet de transmettre la métrique MOS Vidéo.

5.1.4 Le protocole RTCP XR adapté

Dans [CLARK et al, 2008], les auteurs proposent une extension du protocole RTCP XR qui définit un rapport de métriques QoE pour l'utilisation dans des services audio, vidéo.

Pour cela ils proposent un nouveau bloc de rapport RTCP XR (Figure 17).

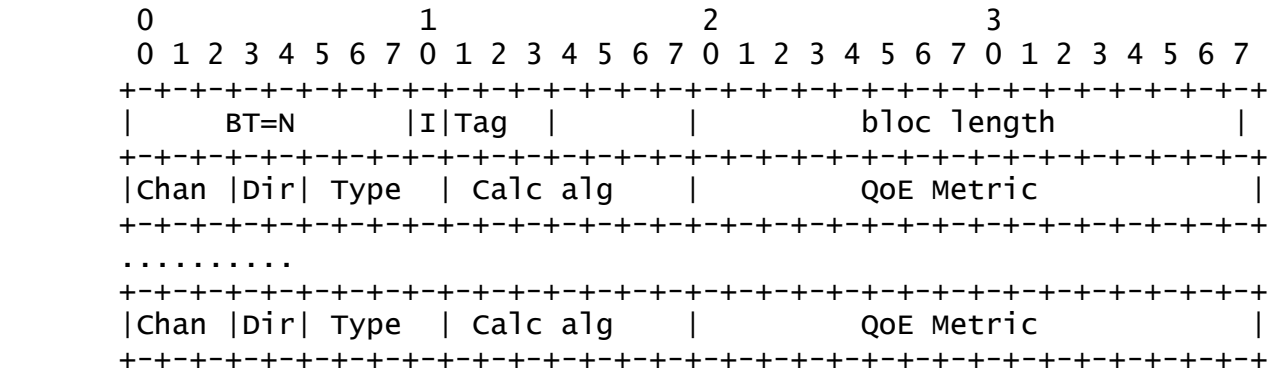


Figure 17 : Structure du bloc de rapport QoE [CLARK et al].

Voici la signification des champs principaux :
BT : type de bloc, RTCP XR bloc type qui doit être fourni par l'IANA quand ce nouveau bloc sera standardisé.
I : *Interval Metric flag*, si I=1 valeur des métriques QoE de type intervalle, si I=0, valeur des métriques QoE de type cumulatif.
Tag : association avec un bloc d'identification de mesure (Measure Identifier block), qui reprend des informations sur les sources (SSRC, …). Dans notre contexte (transmission unicast), ce bloc n'est pas absolument nécessaire.
Channel : le numéro de canal du flux vidéo ou audio auquel la métrique s'applique.
Type : type de métrique, par exemple 0100 = Video Quality MOS (MOS-V).
Calc alg : numéro de l'algorithme de calcul, pour l'instant il n'y a que trois algorithmes définis.
QoE *Metric* : valeur de la métrique QoE calculée, en représentation en échelle entier 8 :8 (8 :8 integer scaled representation). Cela permet des valeurs entre 0,0 à 255,996 avec une précision de 1/256 = 0,00390625.

Un bloc de rapport QoE peut contenir plusieurs valeurs de métriques pour différents flux ou types de flux (vidéo, audio,…).

Ce bloc de rapport QoE est intégré dans un paquet RTCP XR et peut être envoyé avec un rapport RTCP RR comme paquet composé.
Dans notre contexte, le bloc de rapport QoE pourrait être utilisé pour renvoyer la métrique MOS Vidéo, bien que ce bloc de rapport ne soit pas encore standardisé.

5.2 Le protocole RTSP

5.2.1 Généralités

Le *Real-Time Streaming Protocol* (RTSP) qui est spécifié dans le RFC 2326 [RFC2326], permet d'établir et de contrôler un flux de vidéo ou audio continu. On peut le comparer à une télécommande pour un lecteur de DVD qui permet de sélectionner, jouer, suspendre, arrêter une vidéo.

RTSP ne transmet pas les données du flux multimédia, ceci est réalisé par des protocoles comme RTP ou d'autres mécanismes de transport utilisés pour transporter un flux média continu.

RTSP définit des méthodes pour allouer et utiliser des ressources de flux sur le serveur. Certaines méthodes doivent être implémentées, d'autres sont recommandées ou optionnelles. Voici quelques méthodes :

-*SETUP* : fait en sorte que le serveur alloue des ressources pour un flux et démarre la session RTSP

-*PLAY and RECORD* : démarrent la transmission des données pour un flux alloué via SETUP.

-PAUSE : arrête temporairement le flux sans libérer les ressources allouées au serveur.

-*TEARDOWN* : libère les ressources associées avec le flux. La session RTSP se termine.

-OPTIONS : retourne les méthodes supportées par le Serveur/Client.

-*GET_PARAMETER* : retourne la valeur d'un paramètre d'un flux spécifié

-*SET-PARAMETER* : demande d'assigner un paramètre d'un flux spécifié à une valeur donnée.

-*DESCRIBE* : renvoie la description d'une présentation demandée et de tous ses flux. La description fournit des informations sur la pile du protocole, codecs, types de RTP dynamiques, …

Le fonctionnement de RTSP se base sur une technique de requête/réponse. Le client (ou le serveur) envoie une requête au serveur qui contient entre autres la méthode RTSP, l'URI du média, les en-têtes,… Le serveur analyse la requête et renvoie une réponse contenant entre autres un code de statut, et les en-têtes qui correspondent à la requête demandée.

RTSP définit également des champs d'en-têtes. L'utilisation de ces champs dépend de la requête/réponse et peut être obligatoire, recommandée ou optionnelle.

Le déroulement d'une session RTSP pour jouer un flux média peut être le suivant (l'Annexe A donne une représentation détaillée d'une session RTSP) :

-le client demande une description du média au serveur via *DESCRIBE* ou une autre technique (HTTP). Le serveur renvoie la description demandée.

-en tenant compte des informations de la description reçue, le client demande d'allouer les ressources pour le flux média au serveur par la méthode *SETUP*. Le serveur confirme.

-le client demande le démarrage du flux par la méthode *PLAY*. Le serveur confirme et commence à transmettre le flux vers le client.

-le client demande d'arrêter le flux par la méthode *TEARDOWN*. Le serveur confirme, arrête la transmission du flux et termine la session RTSP.

Le RFC2326 ne propose pas directement la possibilité de renvoyer la métrique MOS Vidéo vers le serveur. Mais on pourra étendre les en-têtes existants avec un nouvel en-tête qui désigne la métrique MOS Vidéo. Le client pourra dès lors utiliser la méthode *SET_PARAMETER* pour assigner le nouvel en-tête avec la valeur du MOS Vidéo. Ceci sortira naturellement du standard RFC.

5.2.2 RTSP adapté 3GPP release 6

Dans [3GPP2006], les auteurs présentent un nouvel en-tête RTSP optionnel pour permettre au client et serveur de négocier quelles métriques QoE le client devra envoyer et la fréquence d'envoi de ces métriques. Cet en-tête s'appelle « *3GPP-QoE-Metrics* ».

Ils présentent six métriques QoE pour l'en-tête :

-Métrique de durée de la corruption : correspond à la durée entre la dernière bonne trame et la prochaine bonne trame. Une bonne trame est une trame dont tous les bits ont été reçus sans erreur, et qui soit est une trame IDR, une trame qui ne référence pas une trame précédente ou référence de bonnes trames précédentes (Voir Section 2.3).

-Métrique de durée du buffering initial : correspond à la durée entre la réception du premier paquet RTP et le démarrage de la vidéo.

-Métrique de durée de rebuffering : correspond à la durée de rebuffering, le rebuffering étant défini comme tout blocage du temps de playback due à tout évènement involontaire au niveau du client.

-Perte successive de paquets RTP : correspond au nombre de paquets RTP perdus successivement.

-Déviation du frame rate : correspond à la déviation du FR par rapport au FR prédéfini.

-Durée de jitter : correspond à la durée du jitter de playback, qui ne correspond pas au *jitter* inter-arrivée du [RFC3550]. Le *jitter* étant défini ici comme la différence absolue entre le temps de playback actuel d'une image et le temps de playback expecté d'une image. Si cette valeur est supérieure à 100 ms on parle de jitter.

Les auteurs proposent également un protocole QoE que le client et le serveur devront supporter. Ce protocole règle comment la négociation devra se faire. La Figure 18 montre une négociation de la métrique QoE.

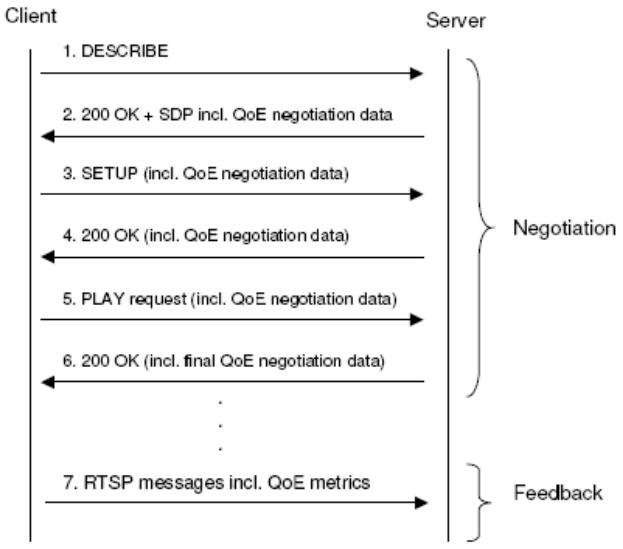


Figure 18 : négociation de la métrique QoE [3GPP2006].

Pour envoyer la métrique vers le serveur, le client peut utiliser la méthode RTSP *SET_PARAMETER* avec l'en-tête de feed-back « *3GPP-QoE-Feedback* ». La fréquence d'envoi de la métrique est définie lors de la négociation.
Dans l'exemple suivant le client renvoie une métrique de durée de la corruption (*Corruption_Duration*) qui contient 2 périodes de corruption de durée 200ms et 1300ms depuis l'envoi de la dernière métrique :

```
C>S     SET_PARAMETER rtsp://example.com/foo/bar/baz.3gp RTSP/1.0
        Cseq: 302
        Session: 17903320
        3GPPQoEFeedback:url="rtsp://example.com/foo/bar/baz.3gp/trackID=3";Corruption_Duration={200
1300}
        Contentlength:0
```

Dans notre contexte, les métriques définies par les auteurs ne permettent pas de renvoyer la métrique MOS Vidéo. Il faudrait ajouter une nouvelle métrique qui correspondra à la métrique MOS Vidéo pour pouvoir utiliser leur nouvel en-tête.

5.3 Choix d'une méthode

Comme on a vu, il n'y a pas de méthode standard pour renvoyer la métrique MOS Vidéo vers le serveur avec les protocoles RTCP/RTSP. En effectuant des adaptations non standard quatre possibilités existent néanmoins pour le feed-back du MOS vidéo :

-Utilisation d'un paquet RTCP *Application defined* : on définit un paquet Application defined avec un nouveau nom et on peut utiliser le champ data pour la valeur du MOS Vidéo. On renvoie le paquet avec le paquet RTCP RR en un paquet composé. La fréquence d'envoi est déterminée par la fréquence d'envoi des paquets RTCP RR [RFC3550].

-Utilisation du RTCP XR adapté avec le type de paquet QoE : ce type de paquet permet de prendre en compte le MOS Vidéo. On renvoie le paquet avec le paquet RTCP RR en un paquet composé. La fréquence d'envoi est déterminée par la fréquence d'envoi des paquets RTCP RR.

-Utilisation du RTSP en ajoutant une nouvelle en-tête/paramètre : on définit un nouvel en-tête pour le MOS Vidéo. Le client peut utiliser la méthode RTSP *SET_PARAMETER* pour mettre à jour cet en-tête avec la valeur MOS Vidéo calculée au niveau du client. La fréquence de mise à jour n'est pas définie et doit être prise en compte par le client.

-Utilisation du RTSP adapté 3GPP release 6 en ajoutant une métrique pour le MOS Vidéo : on définit une nouvelle métrique QoE pour l'en-tête pour prendre en charge le MOS Vidéo. Une négociation au début de la session RTSP permet au client et au serveur de fixer la métrique QoE et la fréquence de mise à jour de celle-ci. Le client met à jour la métrique par la méthode *SET_PARAMETER*.

Pour pouvoir choisir une de ces possibilités, il faut prendre en compte plusieurs critères :

-la compatibilité avec le client média

-la simplicité d'implémentation, vu qu'on se limite dans notre contexte au client, et que la partie serveur n'est pas encore connue.

-la relation avec le MOS Vidéo (Voir Tableau 13).

Le paquet RTCP *Application defined* se base sur le standard RTCP de base qui est supporté par de nombreux clients média. Comme il s'agit ici seulement d'adapter le format d'un paquet RTCP, l'utilisation d'un paquet RTCP *Application defined* pourra être facilement implémentée. Il s'agit d'une méthode utilisée surtout pour l'expérimentation de nouveaux types de paquet RTCP, donc elle n'est pas spécifique au MOS Vidéo.

L'utilisation du RTCP XR adapté avec le type de paquet QoE se base sur les standards RTCP XR et RTCP. Bien que le standard RTCP XR est beaucoup moins répandu au niveau des clients medias, il faut noter que le RTCP XR définit principalement des formats de paquets de rapport et donc une implémentation du type de paquet QoE pourra être réalisée facilement sans devoir implémenter tout le standard RTCP XR. Cette méthode a été spécialement proposée pour le renvoi de métriques QoE, et entre autres le MOS Vidéo.

L'utilisation de RTSP avec un nouvel en-tête se base sur le RTSP de base, qui est également supporté par beaucoup de clients média. Comme le RTSP utilise une technique de requête/réponse, l'envoi du MOS Vidéo à partir du client nécessite une réponse du serveur. Il faudra donc définir un nouvel en-tête pour le MOS Vidéo et adapter la communication requête/réponse sur le client et le serveur. Cette technique utilise une méthode générique pour assigner des valeurs à un en-tête.

L'utilisation du RTSP adapté 3GPP release 6, en ajoutant une métrique pour le MOS Vidéo, se base sur le 3GPP release 6 et l'en-tête « *3GPP-QoE-Metric* », qui est moins supporté par les clients médias.
L'implémentation sera donc plus compliquée. Cette méthode a été spécialement développée pour le renvoi de métriques QoE mais pas spécialement le MOS vidéo calculé au niveau du client.

Protocole	Paquet	Feed-back du MOS
RTCP	RR (PT = 201)	Non, pas de champ disponible
	APP (PT = 204)	Oui, champs « name » + « data »
	XR (PT = 207)	Non, pas de BT « MOS » dans le RFC 3611
	XR adapté [CLARK et al, 2008]	Oui, ajout d'un BT « QoE »
RTSP	Base [RFC 2326]	Peut-être, via « SET_PARAMETER »
	En-têtes « 3GPP_QoE_Metrics » et « 3GPP_QoE_Feedback »	Oui

Tableau 13 : implémentation du Feed-back du MOS dans les paquets.

L'utilisation d'un paquet RTCP XR adapté avec le type de paquet QoE de [CLARK et al, 2008] correspond le mieux aux critères de sélection. On choisit donc cette méthode pour effectuer le feed-back de la métrique MOS.

5.4 Critique de la méthode de feed-back choisie

Dans [OTT et al, 2010], les auteurs présentent des lignes directrices pour étendre le protocole RTCP. Les auteurs précisent que RTCP est un protocole qui utilise normalement un transport peu fiable, *best-effort* et les rapports RTCP devraient être conçus pour un tel environnement en les considérant "pour information".
Les informations renvoyées via le canal de contrôle sont souvent des valeurs moyennées sur une période, et la fréquence des rapports est de l'ordre de plusieurs secondes. En plus une perte des paquets RTCP est possible, donc il n'y a pas de garantie d'un feed-back.
Différentes extensions de RTCP ont essayé d'en faire un protocole de signalisation fiable, instantané ou un canal de commande, ce qui n'est pas le but de RTCP.
Les auteurs remarquent que le feed-back RTCP est insuffisant pour réaliser un contrôle de congestion, mais qu'il est possible de réaliser une adaptation de la bande passante sur une échelle de temps plus grande.
Ils évoquent le fait que le protocole RTCP est destiné à une communication multicast, et que des problèmes peuvent survenir lorsqu'une extension de RTCP est seulement définie pour un cas unicast.
Les auteurs remarquent enfin qu'un nouveau type de bloc RTCP XR est approprié pour transporter des nouvelles métriques concernant la qualité de réception.

L'utilisation du RTCP XR adapté avec le type de paquet QoE de [CLARK et al, 2008] se base sur un nouveau type de bloc RTCP XR pour renvoyer le MOS, qui peut être considéré comme

une métrique concernant la qualité de réception. Ce qui correspond aux lignes directrices présentées.

L'utilisation du protocole RTCP n'est pas destinée aux cas nécessitant un feed-back immédiat et fiable, comme c'est le cas de l'adaptation de la qualité vidéo lors des variations brusques de la bande passante.

Dans le contexte de ce mémoire, la mesure de la qualité vidéo, qui va déterminer la valeur de la métrique renvoyée au serveur, se base sur l'utilisation de modèles VQM. Une caractéristique principale de ces modèles est de calculer une métrique qui a une très bonne corrélation avec le MOS, ce qui peut être réalisé en se basant sur les métriques de l'image, qui utilisent les informations de la vidéo décodée.
Le MOS est une valeur moyennée des opinions d'un groupe de personnes sur la qualité vidéo d'une séquence vidéo d'une certaine durée.
Donc la métrique calculée par les modèles VQM à un moment donné sera une valeur qui prend en compte la qualité vidéo d'une certaine période de la séquence vidéo et pas seulement d'un instant précis.
En plus, le décodage de la vidéo nécessaire aux modèles VQM, qui se basent sur une métrique de l'image, ajoute un retard de traitement dans le feed-back de la métrique.

Donc, comme il ne s'agit pas d'implémenter un feed-back par paquet ou instantané complètement fiable, le feed-back occasionnel proposé par RTCP suffit pour réaliser une adaptation de la bande passante sur une échelle de temps de plusieurs secondes.

Dans l'étude effectuée pour ce mémoire, on s'est limité à un cas de streaming unicast. Dans cette optique on a simplifié la méthode proposée par [CLARK et al, 2008] en supprimant le bloc d'identification de mesure. Ceci ne correspond donc pas aux lignes directrices discutées dans [OTT et al, 2010].
Ce point devra être adapté lors d'une utilisation de ce mémoire pour une réalisation éventuelle future de la partie serveur du système d'adaptation de la qualité vidéo.

6 Adaptation de la qualité vidéo pour le streaming

Les techniques existantes d'adaptation de la qualité vidéo pour le streaming se basent principalement sur une adaptation de la qualité vidéo qui se réalise au niveau du serveur de streaming. Comme on se limite à la partie client dans ce mémoire, l'objet de ce chapitre est de donner un aperçu de ces techniques et non d'effectuer un choix d'une technique spécifique.

6.1 Généralités

On peut d'abord se poser la question de la signification de l'adaptation de la qualité vidéo pour le streaming dans notre contexte. On a vu que le streaming vidéo sur terminaux mobiles peut subir une bande passante changeante, qui peut conduire à des pertes de paquets. Ceci va influencer fortement la qualité vidéo affichée au terminal client.

Donc dans notre contexte, l'adaptation de la qualité vidéo va principalement consister à adapter le débit binaire (BR) nécessaire de la vidéo envoyée vers le terminal client mobile, en se basant sur une métrique de qualité calculée au niveau du client mobile.
La figure 19 montre un schéma d'un système de streaming adaptatif générique. Il faut noter que la génération du feed-back peut également se faire après le décodeur (métrique d'image).

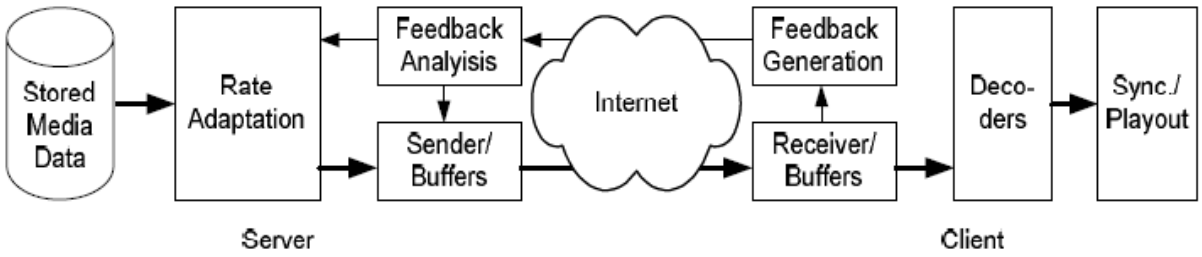


Figure 19 : Aperçu d'un système de streaming adaptatif. [SCHIERL et al, 2004]

6.2 Techniques d'adaptation du débit binaire (bit rate) vidéo

On va passer en revue quelques techniques utilisées pour adapter le débit binaire (bit rate) de la vidéo au niveau du serveur. Ces techniques peuvent être catégorisées suivant deux axes :

-axe spatial : concerne le codage intra –image

-axe temporel : concerne l'inter-image

Les concepts suivants seront traités :

-Multi Bit Rate (MBR) Streaming (axe spatial).

-Ne pas transmettre les images non de référence (*frame dropping*) (axe temporel)

-Transcodage de la vidéo avec un débit binaire réduit (axe spatial et/ou temporel)

On peut constater qu'on retrouve souvent des concepts similaires avec des appellations différentes.

Pour information, on présente également une technique MBR qui se base sur le HTTP Live streaming de Apple (donc elle n'utilise pas le protocole RTP/RTSP/RTCP), qui propose un système où c'est le client qui choisit la version de la vidéo.

6.2.1 *Bit-Stream Switching* (BSS)

Dans [SCHIERL et al, 2004], les auteurs présentent une technique MBR d'adaptation du débit binaire spécifiquement pour le codec H.264/AVC.

On dispose de plusieurs versions de la vidéo encodées avec des débits binaires différentes (MBR Streaming). Chaque version contient des trames '*Instantaneous Decoder Refresh* (IDR)' qui sont insérées régulièrement dans la vidéo. Lorsqu'on arrive à une trame IDR, on peut passer à une version différente (avec un débit binaire moindre) de la vidéo sans interruption.

Avec ce système on diminue la qualité générale de la vidéo transmise, mais le débit binaire réduit de la vidéo permet de recevoir la vidéo au niveau du client sans pertes de paquets due à la bande passante réduite. On peut également noter que la fréquence d'image (FR) reste la même. Le passage ne peut se faire lors d'une trame IDR, donc ce système est dépendant du nombre de trames IDR dans la vidéo. La figure 20 montre le passage d'une version de la vidéo vers une autre.

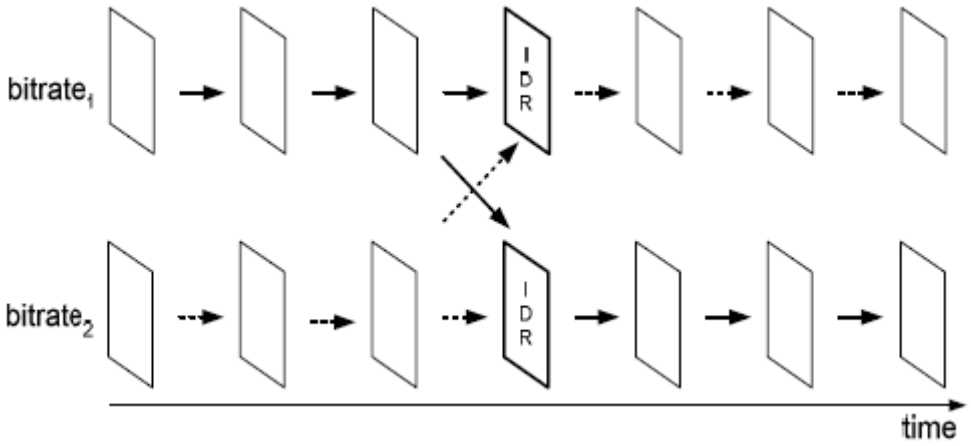


Figure 20: *Bit-Stream switching* avec H.264/AVC. [SCHIERL et al, 2004]

[WEN et al, 2006] remarquent que le passage vers une version différente devra se faire au niveau d'une trame Intra–codée (trames I ou IDR). Pour des raisons d'efficacité de codage ces trames i ou IDR sont éloignées les unes des autres, ce qui limite le passage d'une version à une autre. Tandis que l'utilisation des trames SP, qui sont des séquences de macroblocks de type P ou I, pour le passage augmentera fortement la complexité.

On retrouve une technologie similaire chez d'autres auteurs :
[BIRNEY2003] discute l'utilisation de l'encodage Multi Bit-Rate pour la technologie Windows Media.
[MULROY et al, 2009] utilisent une technique similaire en conjonction avec leur protocole MULTCP.

6.2.2 *Temporal Scalability* (Ts)

Une autre technique présentée par [SCHIERL et al, 2004] est le *temporal scalability* (TS) du H.264. Une vidéo H.264 contient des trames qui ne servent pas comme trames de références pour d'autres trames. L'idée est que si on ne transmet pas ces trames (on diminue le débit binaire), le décodeur du client peut quand même décoder les autres trames de la vidéo.
Le standard H.264 permet une représentation en niveau des images codées. On peut par exemple retrouver dans le niveau de base les trames IDR, dans le premier niveau d'amélioration (*enhancement level*) on retrouve les trames qui serveent comme trame de référence à d'autres trames (*referenced frames*) et dans le deuxième niveau d'amélioration on a les trames qui ne sont pas utilisées comme trames de référence (*non referenced frames*). On peut maintenant transmettre seulement le niveau de base et le premier niveau (diminution du débit binaire) sans que la qualité au niveau du client ne se dégrade trop.

La Figure 21 montre l'effet sur le PSNR lors de pertes de trames de référence et des trames qui ne servent pas comme référence. On peut constater que la qualité vidéo est beaucoup moins dégradée lors de pertes de trames non de référence.

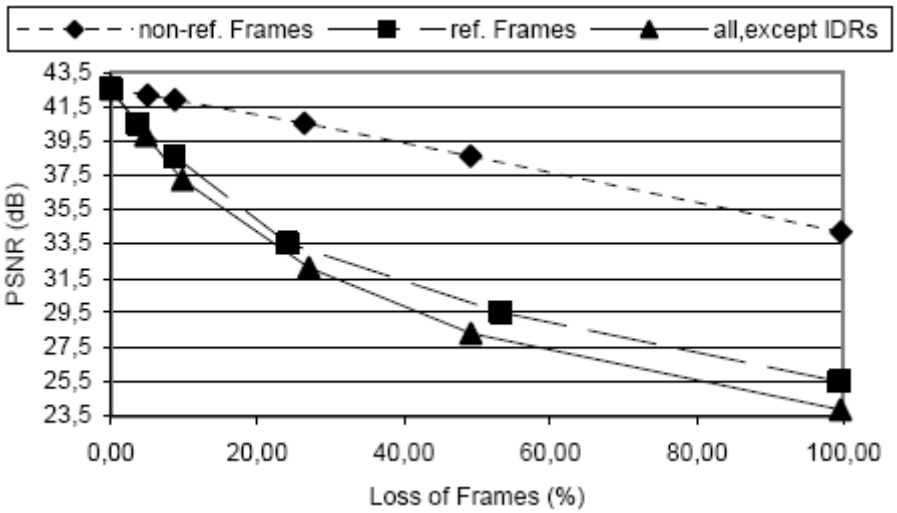


Figure 21 : Diminution du PSNR en fonction de pertes d'images de type différent. [SCHIERL et al, 2004]

D'autres auteurs parlent d'une technique similaire :

[BIRNEY2003] discute le "*stream thining* ", la diminution de la fréquence d'images (FR) afin de diminuer le débit binaire.

On peut remarquer que comme on ne transmet pas toutes les trames, le nombre de trames par seconde (FPS) reçues et affichées par le client va diminuer.

6.2.3 Transcodage de la vidéo

Dans [Jammeh et al, 2002], les auteurs décrivent un système qui utilise un transcodeur temps réel pour adapter le débit binaire de la vidéo (Figure 22).
En fonction du feed-back du client, le transcodeur encode la vidéo originale au BR demandé. Ceci a comme avantage de n'utiliser qu'une version de la vidéo et que le débit binaire peut prendre plus de valeurs que pour la technique MBR.
L'inconvénient réside dans la performance nécessaire au transcodage.

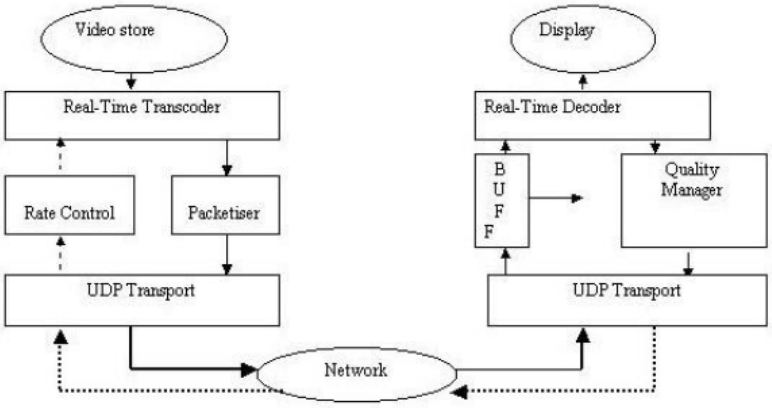


Figure 22 : Adaptation du débit binaire par transcodage. [Jammeh et al, 2002]

6.2.4 Les flux alternatifs (*Stream Alternates*)

Dans [APPLE2010], les auteurs décrivent une technique qui se base sur HTTP *Live streaming* de Apple, qui permet au client de passer dynamiquement à une version différente (débit binaire réduit) de la vidéo lorsque la bande passante diminue. (Donc on n'utilise pas les protocoles RTP/RTSP/RTCP.)
Le client utilise un fichier d'indexation (*Index file*) pour trouver les flux alternatifs pour la vidéo. Suivant les auteurs le client utilise des heuristiques basées sur des mesures du débit du réseau pour déterminer l'instant du passage d'une version à une autre (Figure 23).

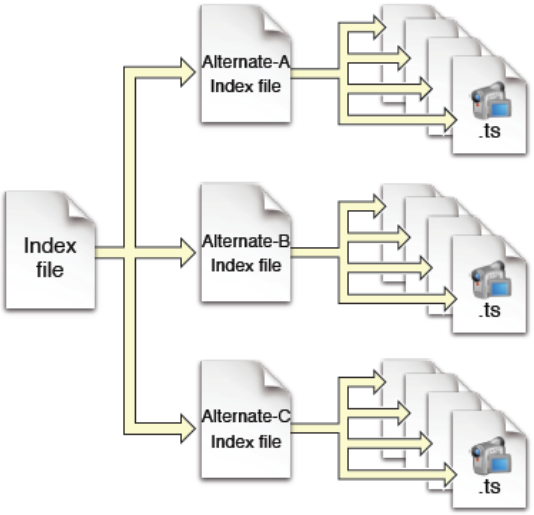


Figure 23 : La structure des flux alternatifs. [APPLE2010]

6.2.5 Conclusion

On a présenté plusieurs techniques d'adaptation de la qualité vidéo pour le streaming vidéo RTP/RTSP/RTCP basées sur une adaptation de la qualité au niveau du serveur. Les techniques visent à réduire le débit binaire nécessaire pour transmettre la séquence vidéo vers le client et se basent sur :

- l'utilisation de plusieurs versions d'une vidéo encodées avec des débits différents *(Bit-Stream Switching)*
- la suppression de certaines trames lors de la transmission de la vidéo (*Temporal Scalability)*
- le transcodage de la vidéo originale avec un débit binaire plus faible

On peut remarquer que certaines techniques peuvent diminuer la fréquence d'images (FPS) au niveau du client, dont il faut tenir compte pour la solution proposée.

7 Présentation de la solution choisie

Les chapitres précédents nous ont permis de choisir différents composants nécessaires pour réaliser la partie client d'un système d'adaptation automatique de la qualité d'un service de streaming vidéo.

L'objectif de ce chapitre est d'intégrer tous les composants et de présenter notre solution générale.
Pour cela, il faut encore choisir le dernier élément, l'application média client qui va être utilisée pour implémenter le système. On va également proposer une adaptation du modèle VQM choisi aux contenus vidéo continus.

7.1 Choix du client média

L'objectif est de définir les critères de sélection du client média, de choisir le client média à utiliser et de présenter ses principales caractéristiques.

7.1.1 Critères de choix

Un objectif de ce mémoire est d'implémenter dans un client média le modèle VQM déterminé et le feed-back MOS vidéo calculé. Donc un premier critère de sélection est que le code source de cette application soit libre (open source) pour pouvoir l'adapter.

Une autre caractéristique est la compatibilité avec les protocoles RTP/RTSP/RTCP. Bien que le feed-back se fasse via le RTXP XR adapté [CLARK et al, 2008], le client ne doit pas nécessairement supporter le protocole RTCP XR vu qu'il s'agit principalement de définitions de formats de types de paquets RTCP. La compatibilité avec le protocole RTSP concerne le contrôle du flux multimédia (établir la session de streaming,…) et non le feed-back MOS vidéo.

Pour faciliter l'implémentation dans le client média, il est également important de disposer d'une très bonne documentation sur le client média, qui en explique les concepts principaux et le fonctionnement interne.

Comme précisé dans l'introduction, dans les limites fixées pour ce mémoire, on a décidé de remplacer le terminal mobile par un PC standard avec le système d'exploitation LINUX (UBUNTU 9.10). La compatibilité de l'application client avec des systèmes d'exploitation utilisés sur des terminaux mobiles sera cependant considérée, en vue d'une éventuelle implémentation future sur un terminal mobile.

7.1.2 VLC

VLC est un lecteur média open source développé par le projet VideoLAN [VIDEOLAN]. VLC dispose de fonctionnalités avancées qui peuvent en plus être étendues par des plugins. Des plugins compatibles avec les protocoles RTP/RTCP/RTSP de base existent. Il est fort répandu avec plus de 400 millions de téléchargements et a été porté sur plusieurs systèmes d'exploitation. Il existe des versions avec des fonctionnalités limitées pour des terminaux mobiles, compatibles avec les systèmes d'exploitation wince et familiar Linux (linux sur smartphone). Le projet VideoLAN propose un site pour développeurs (http://wiki.videolan.org/Developers_Corner), qui fournit une documentation de base pour la programmation de l'application.

7.1.3 Le framework GStreamer

GStreamer est un framework multimédia open source qui permet de développer facilement des applications multimédia [TAYMANS et al, 2010]. GStreamer se base sur une architecture de plugins, ce qui permet d'étendre ses fonctionnalités [BOULTON et al, 2010]. Des plugins compatibles avec les protocoles RTP/RTCP/RTSP de base existent. Le framework a été porté sur plusieurs systèmes d'exploitation, ainsi que sur des terminaux mobiles (Nokia N70, version limitée pour Symbian OS,…). Sur le site de GStreamer (http://gstreamer.org/), on trouve une très bonne documentation pour le développement de plugins et d'applications GStreamer.

7.1.4 Comparatif des clients média

Le Tableau 14 compare les deux clients média en tenant compte des critères définis.

Client média	Code source libre	Compatible RTP/RTCP/RTSP de base	Documentation	Compatible avec terminal mobile
VLC	oui	oui \via plugin	limitée	fonctions limitées
GStreamer	oui	oui \ via plugin	détaillée	fonctions limitées

Tableau 14 : Comparaison de VLC et GStreamer.

Aucun des deux clients ne remplit complètement tous les critères définis. Ceci est surtout dû à une compatibilité limitée avec les terminaux mobiles.
Néanmoins, l'utilisation du framework GStreamer est plus avantageuse vu qu'il dispose d'une très bonne documentation détaillée.

On choisit donc GStreamer comme client média à adapter.

7.1.5 Introduction au framework GStreamer

Dans [TAYMANS et al, 2010], les auteurs présentent GStreamer. GStreamer est un framework qui permet de développer des applications multimédia de tout type. Le framework est basé sur des plugins qui proposent des fonctionnalités de codec, filtres, … et sur une partie de base (core framework) qui fournit des services pour les plugins, les flux de données,…

La Figure 24 montre un aperçu général du framework GStreamer.

GStreamer fournit entre autres :

-une API pour les applications multimédia

-une architecture de plugin

-une architecture pour les pipelines

-un mécanisme pour la négociation et le traitement des types de média

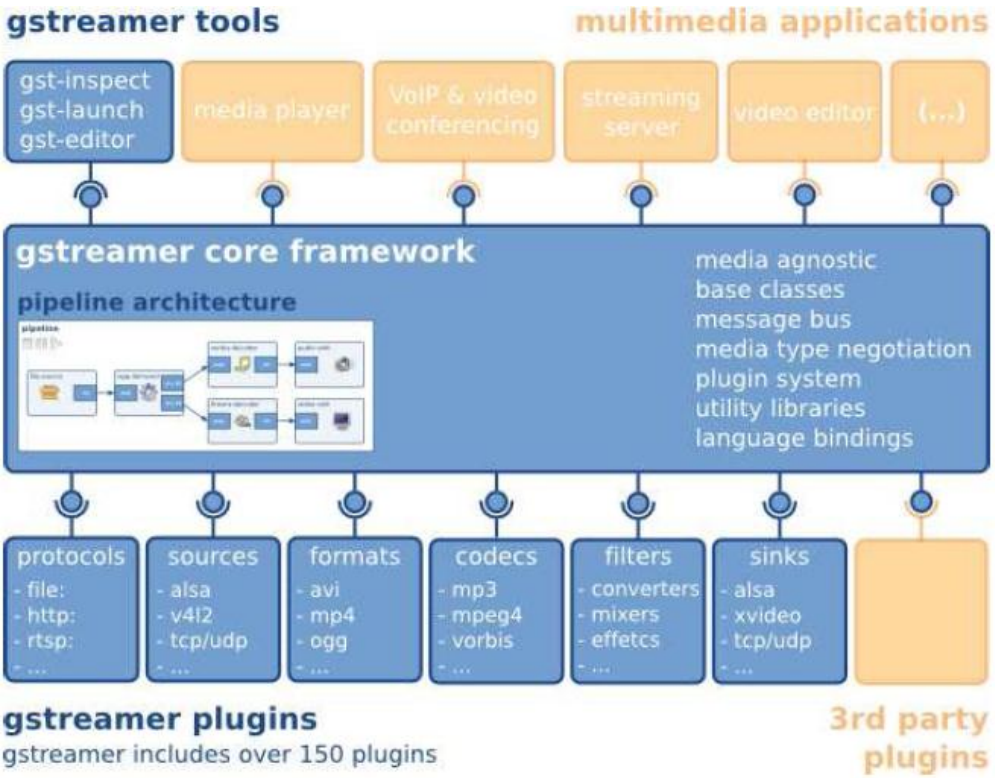


Figure 24 : Aperçu général du framework GStreamer [TAYMANS et al].

Un concept principal de GStreamer est celui de pipeline média. Un pipeline est constitué d'éléments GStreamer, chaque élément ayant une fonction spécifique, qui sont connectés entre eux pour former une chaîne. Un plugin contient un ou plusieurs éléments, et permet à GStreamer d'utiliser ces éléments. Le pipeline définit le sens du flux des données entre les éléments et peut ainsi réaliser une certaine tâche, comme l'affichage d'une vidéo. La Figure 25 représente un pipeline d'un lecteur ogg.

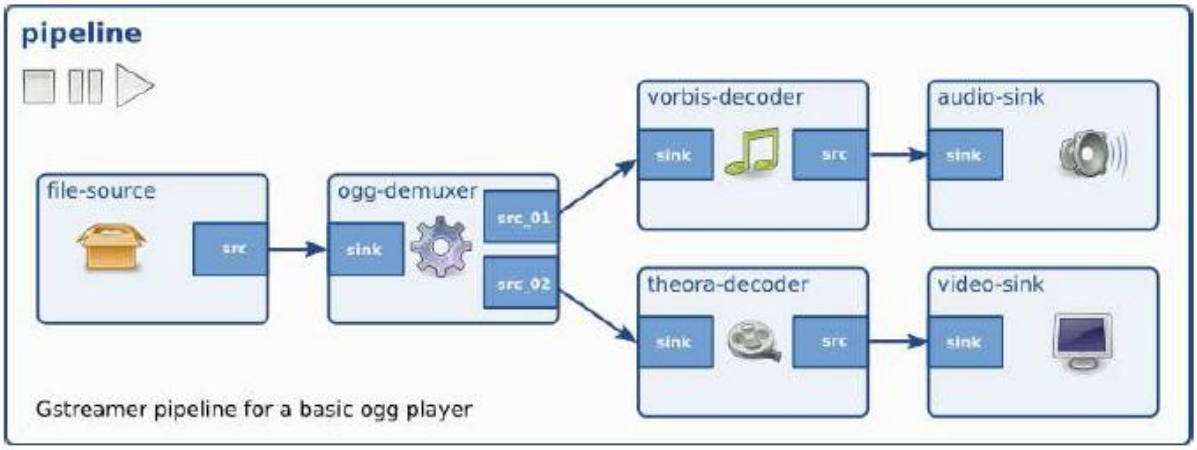


Figure 25 : représentation d'un pipeline média avec ces éléments GStreamer [TAYMANS et al].

Les pads sont les entrées et sorties des éléments, qui permettent de connecter les éléments entre eux. Les données de streaming (*buffers*) transitent d'un pad source (src) d'un élément, vers un pad sink d'un autre élément, donc ils transitent toujours vers l'aval (*downstream*). Les bins représentent des conteneurs (*containers*) pour une collection d'éléments. Un pipeline est un sous-type d'un bin.

Un élément GStreamerpeut avoir plusieurs états :

-NULL : état par défaut, pas de ressources allouées

-*READY* : les ressources globales sont allouées.

-*PAUSED* : le stream est ouvert, l'élément peut se préparer à la lecture (traitement des données), mais l'horloge ne tourne pas, donc pas de transfert de buffers d'un élément vers un autre.

-*PLAYING* : idem *PAUSED*, mais l'horloge tourne et le buffer du stream est passé d'un élément vers une autre.

GStreamer dispose d'un bus, qui permet la communication entre les éléments ou entre les éléments et l'application. Plusieurs mécanismes sont proposés :

-évènements (*events*) : ce sont des objets envoyés entre éléments, ou de l'application vers des éléments. Ils peuvent être envoyés vers l'aval ou vers l'amont. Ils peuvent être envoyés soit en synchrone avec les buffers de streaming soit en asychrone. Ils sont utilisés pour renseigner sur l'état du flux,…

-messages : ce sont des objets envoyés des éléments via le bus vers l'application (End-of-stream notification, Erreurs,…).

-requêtes (*queries*) : permettent à l'application et aux éléments de demander des informations au pipeline. Ils peuvent être envoyés vers l'aval ou vers l'amont.

La Figure 26 représente les différents mécanismes de communication.

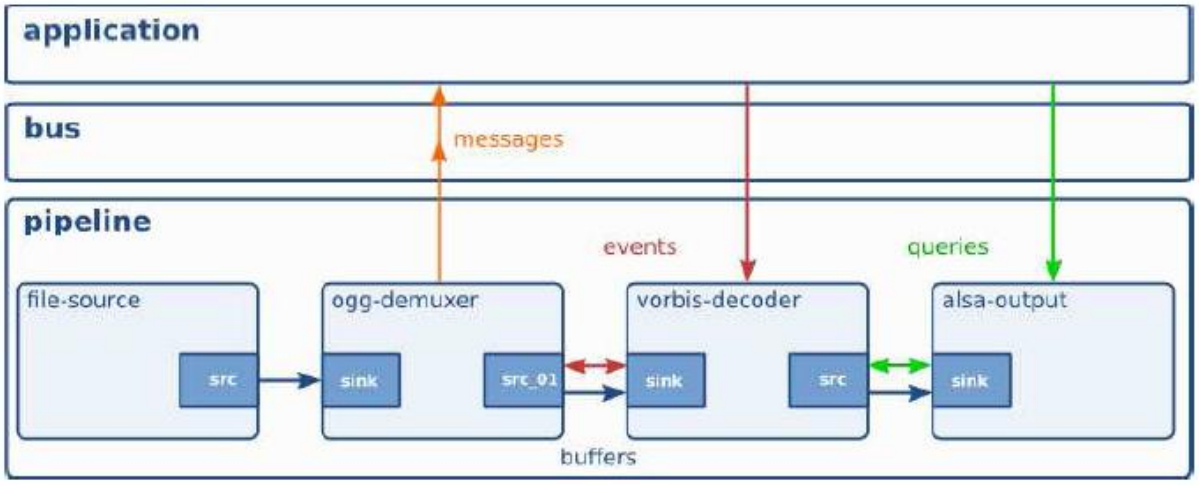


Figure 26 : Mécanismes de communication via le bus GStreamer [TAYMANS et al, 2010].

GStreamer se base sur une horloge pour synchroniser le pipeline. Cette horloge est soit fournie par un fournisseur d'horloge (un élément GStreamer du pipeline), soit l'horloge système. Le fournisseur doit assurer que le temps d'horloge représente au mieux le temps média courant. Plusieurs références de temps sont calculées à partir du temps de l'horloge (clock-time), voir Figure 27 :

-temps de base (*base time*) : temps de l'horloge lors du lancement de la lecture (play)

-temps de lecture (*running time*) : temps de l'horloge – temps de base.

-temps de streaming : temps qui indique la position dans le média.

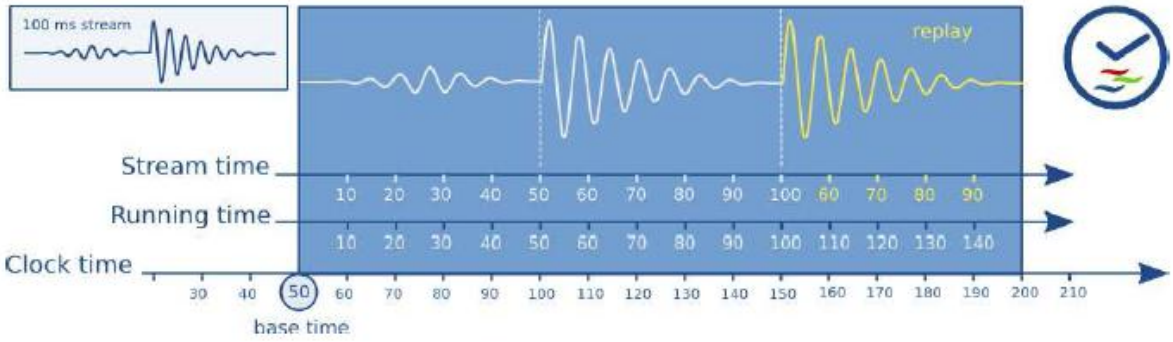


Figure 27 : Les différents temps de GStreamer [TAYMANS et al]. Suite à un replay, le streaming time passe de 100 à 60 ms, pendant que le running time poursuit sa course.

7.2 Adaptation du VQM basé sur la rupture de la fluidité vidéo à la vidéo continue

Le modèle VQM basé sur la rupture de la fluidité proposé par [PASTRANA2006], [PASTRANA2005] et [PASTRANA2007] a initialement été utilisé pour déterminer le MOS vidéo de séquences de vidéo de 10 secondes de durée. Les auteurs ont également utilisé une fenêtre d'analyse d'une durée égale à la longueur de la séquence (10 sec) pour calculer la distribution des durées des dégradations. La durée des séquences vidéo a été choisie pour

éviter l'effet oubli (après un certain temps, une dégradation est considérée comme moins grave).

Dans notre contexte, deux éléments doivent être pris en considération:

-on dispose de contenus vidéo continus de durée variable, non des séquences de 10-15 secondes.

-la fréquence de feed-back de la métrique MOS vidéo est déterminée par la fréquence d'envoi des paquets RTCP RR et est donc égale à quelques secondes.

Donc on doit adapter le modèle VQM en conséquence.

Une idée pour prendre en compte le caractère continu de la vidéo et la fréquence de feed-back, serait de pouvoir calculer la métrique non seulement à la fin d'une séquence, mais également à n'importe quel moment durant la lecture du flux vidéo. Une idée possible est d'utiliser une fenêtre d'analyse de 10 secondes, comme les auteurs, mais qui se déplace avec le temps. La fenêtre d'analyse correspondra aux 10s précédant le moment pour lequel on veut calculer le MOS. On enregistre temporairement les dégradations avec leurs informations sur la durée et leur instant de démarrage. Ces informations permettront de déterminer si une dégradation enregistrée temporairement se trouve dans la fenêtre d'analyse et doit donc être pris en compte pour le calcul MOS.

Prenons par exemple un temps t1 qui correspond à un certain moment pendant la lecture du flux vidéo (Figure 28). Si on veut connaître le MOS vidéo à ce temps t1 durant la lecture du flux vidéo, on calcule la métrique sur base de la distribution des durées des dégradations, qui commencent ou se terminent, sur la période (t1 – 10 s) jusque t1. On aura toujours une fenêtre d'analyse de 10 s, mais qui prend en compte les dégradations des derniers 10s avant le moment t1.

A un moment ultérieur dans la lecture du média, t2, la fenêtre d'analyse commence à (t2 - 10s) et se termine au temps t2.

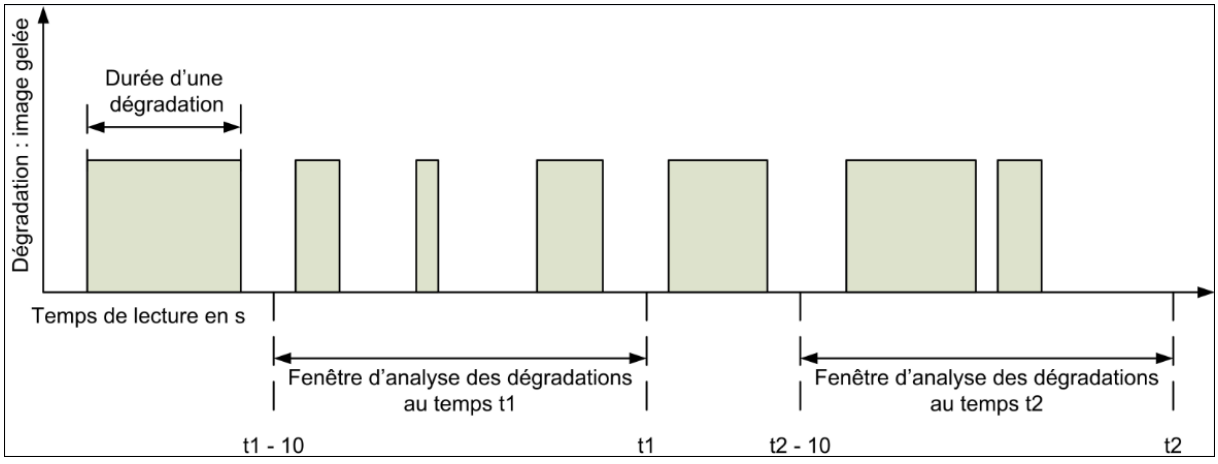


Figure 28 : La fenêtre d'analyse des dégradations qui se déplace.

Une question qu'on peut se poser est, si l'adaptation d'une métrique VQM vers de la vidéo continue est pertinente ?

En effet, l'objectif d'une métrique VQM est de calculer une valeur qui se rapproche le plus possible du MOS vidéo. Mais ce MOS vidéo a été obtenu par des évaluations subjectives de la qualité vidéo qui utilisent des séquences vidéo limitées à 10-15s de longueur. Donc le domaine où les valeurs des métriques VQM sont validées se limite aux séquences vidéo d'une longueur de 10-15s. Pour les vidéos de plus longue durée, les valeurs des métriques ne sont pas validées.

Dans notre cas d'utilisation, il ne s'agit pas de calculer une métrique pour toute la vidéo, mais de pouvoir calculer la métrique à un certain moment. Dans ce cas, l'utilisation de la métrique VQM basée sur la rupture de la fluidité, adaptée avec une fenêtre d'analyse qui se déplace, se limite à la longueur de la fenêtre d'analyse, qui correspond normalement à 10s. Cela correspond aux conditions précitées.

7.3 L'architecture logique de la solution

On dispose maintenant de tous les éléments pour proposer une solution de la partie client d'un système d'adaptation automatique de la qualité vidéo via RTP/RTSP/RTCP.

On a déterminé un modèle VQM pour pouvoir calculer une métrique de qualité vidéo, le MOS vidéo : Le modèle VQM basé sur la rupture de la fluidité proposé par [PASTRANA2006], [PASTRANA2005] et [PASTRANA2007] qu'on a adapté à la vidéo continue.

On a choisi une extension du protocole RTCP XR avec le nouveau type de paquet QoE [CLARK et al, 2008] pour effectuer le feed-back de cette métrique vers le serveur média.

Finalement on a choisi le framework GStreamer comme client média à adapter.

La Figure 29 montre le schéma de la solution proposée.

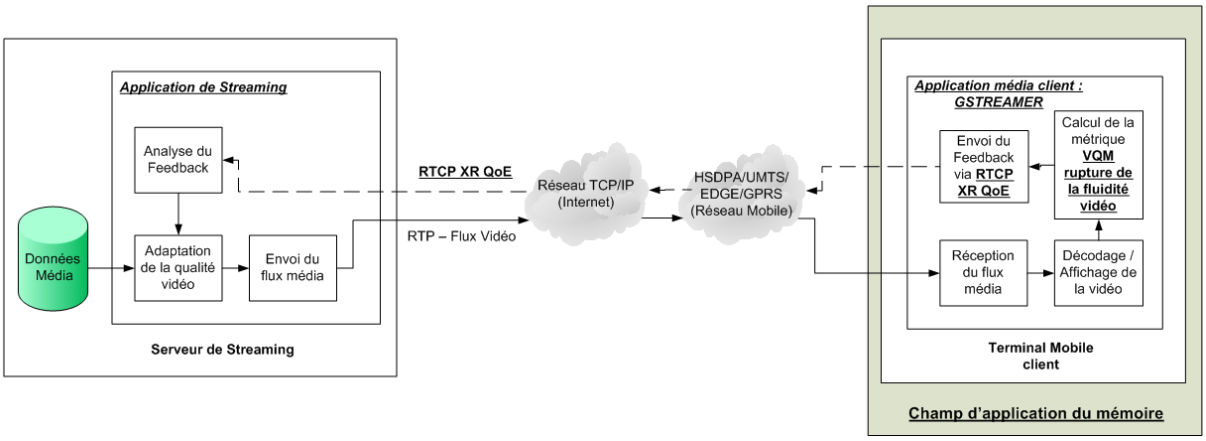


Figure 29 : Schéma de la solution proposée.

Deux fonctionnalités doivent être ajoutées au client média :

-le calcul de la métrique MOS

-le feed-back de cette métrique via RTCP XR QoE

Comme on utilise le framework GStreamer, on peut soit développer un nouveau plugin avec un élément, soit adapter un élément d'un plugin existant.
Pour calculer la métrique MOS, on doit disposer de la vidéo décodée. Si on veut utiliser un élément existant à adapter, celui-ci doit être capable de traiter les données vidéo en entrée. Celui-ci sera donc un élément de type filtre ou un élément de sortie vidéo (affichage vidéo). Ces éléments ont normalement des fonctionnalités spécifiques appartenant à un certain domaine (affichage vidéo, filtre couleurs, transcodage RGB-YUV,…), qui est hors du domaine du calcul des métriques de qualité vidéo. L'ajout d'une fonctionnalité de calcul MOS à ces éléments n'est donc pas intéressant. On va développer un nouveau plugin avec un élément pour calculer le MOS, qu'on appelle : " NRVQM ".

En ce qui concerne le feed-back du MOS via RTCP XR QoE, il existe déjà une série d'éléments qui permettent d'utiliser les protocoles RTP/RTCP/RTSP de base. Comme le développement d'un nouvel élément qui réalise ces fonctionnalités est beaucoup trop coûteux, on va adapter les éléments existants pour réaliser le feed-back MOS via RTCP XR QoE.
La Figure 30 montre les plugins avec les éléments développés/adaptés pour le framework GStreamer dans le cadre de ce mémoire. Le code source est repris en Annexe B et Annexe C.
Le framework GStreamer ne peut fonctionner à lui tout seul. Il faut une application qui construit et contrôle le pipeline GStreamer. Il y a un utilitaire GStreamer 'gst-launch' qui fonctionne en ligne de commande et permet facilement de construire/contrôler un pipeline GStreamer.

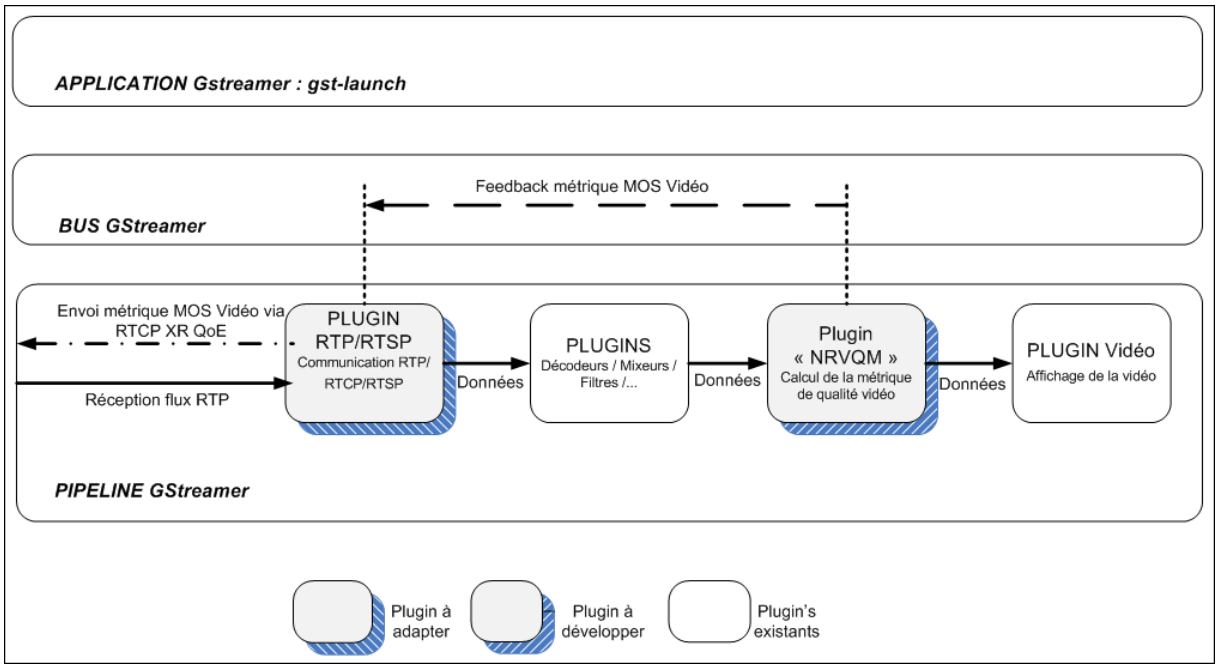


Figure 30: Utilisation du framework GStreamer pour la solution proposée.

8 Implémentation

L'objectif de ce chapitre est de déterminer comment on peut implémenter la solution proposée dans le framework GStreamer.

On va d'abord déterminer l'implémentation du nouveau plugin avec un élément GStreamer " NRVQM " qui aura pour fonctionnalité de calculer la métrique MOS sur base du modèle VQM, basé sur la rupture de la fluidité proposée par [PASTRANA2006], [PASTRANA2005] et [PASTRANA2007] qu'on a adapté à la vidéo avec un contenu continu. Une autre fonctionnalité de cet élément est d'envoyer sur le bus GStreamer la valeur de la métrique MOS calculée, qui sera interceptée par les éléments correspondants (élément GstRTSPSrc, …).
Puis on va déterminer comment on peut adapter les éléments des plugins RTP/RTCP/RTSP existants pour implémenter le feed-back de la métrique MOS via le protocole RTCP XR QoE [CLARK et al, 2008].

8.1 Le plugin/élément « NRVQM »

8.1.1 Généralités

Les plugins sont programmés avec le langage C. GStreamer utilise le modèle de programmation GObject (http://library.gnome.org/devel/gobject/unstable/index.html), qui constitue un framework orienté objet pour le langage C.

La hiérarchie de l'élément Gst de base par rapport au GObject peut se représenter comme :

```
GObject
        +---- GstObject
                +----GstElement
```

Un plugin GStreamer contient les parties suivantes [BOULTON et al, 2010]:

-définitions des structures de données
-définitions des classes pour l'élément inclus
-des fonctions d'initialisation de la base, la classe et l'instance d'un élément
-une fonction d'initialisation du plugin (avec enregistrement des éléments du plugin)
-des fonctions de spécification des pads
-des fonctions de spécification des capacités des pads
-une fonction de chaine, qui traite les données du flux
-des fonctions pour traiter les états de l'élément
-des fonctions pour ajouter et traiter les propriétés d'un élément
-des fonctions pour traiter les signaux (évènements, messages,…)

Si la classe de l'élément contenu dans le plugin hérite ses propriétés d'une classe père, certaines de ces fonctions et définitions peuvent être fournies par la classe père, et cette classe père peut fournir d'autres fonctions.

Pour développer un plugin GStreamer, on peut soit créer un plugin complètement nouveau, en mettant la structure, les définitions et les fonctions de base nécessaires à un plugin GStreamer, et ajouter l'élément avec les fonctionnalités demandées, soit on peut se baser sur des templates de plugins existants et y ajouter l'élément avec les fonctionnalités demandées. Ceci est souvent plus facile comme toute la structure de base est déjà définie par le template.

Pour choisir un template à utiliser il faut déterminer les caractéristiques nécessaires de l'élément du nouveau plugin.
L'élément NRVQM devra avoir un pad sink d'entrée pour recevoir les données vidéo et un pad source de sortie pour transmettre la vidéo à l'élément suivant. On calcule la métrique MOS sur base des données vidéo reçues, mais on ne change pas les données vidéo avant des les transmettre à l'élément suivant. On peut se limiter à la vidéo en format YUV. Dans le cas d'une vidéo au format RGB, un élément de conversion RGB vers YUV pourra simplement précéder l'élément NRVQM.

Ces caractéristiques sont remplies par le template « gstvideotemplate » qui crée un plugin pour des éléments de type filtre, un élément filtre étant un élément qui traite un flux de données. En plus ce template implémente directement les fonctionnalités suivantes (via les classes GstVideoFilter et GstBaseTransform):

-implémente un pad d'entrée (*sink*) et un pad de sortie (source)
-traite les changements d'états
-permet de laisser tomber des trames (*frame dropping*)
-utilise par défaut les mêmes formats de données sur les deux pads
-implémente les définitions et fonctions d'initialisation
-configure la spécification et l'assignation des capacités des pads

L'hiérarchie des objets étant :

```
GObject
        +---- GstObject
                +----GstElement
                        +----GstBaseTRansform
                                +----GstVideoFilter
```

On présente quelques fonctions importantes créées automatiquement par le template dans le fichier gstnrvqm.c (voir Annexe B.1):
Gst_nrvqm_init (GstNrvqm * nrvqm, GstNrvqmclass * g_class)
nrvqm étant un pointeur sur un objet de type GstNrvqm, g_class étant un pointeur sur la classe GstNrvqmClass. Cette fonction permet entre autres d'initialiser les propriétés et les structures de données et d'assigner les fonctions de traitement des évènements.

gst_nrvqm_transform_ip (GstBaseTransform * base, GstBuffer * outbuf)
base étant un pointer sur un objet GstBaseTransform, outbuf étant un pointeur sur un objet GstBuffer (qui contient entre autre les données d'une image vidéo). Cette fonction permet de traiter les données (images vidéo) reçues par l'élément nrvqm.

Gst_nrvqm_set_caps (…), permet d'assigner les capacités des pads. Ici le format d'image vidéo est fixé à YUV 4:2:0 (Voir Section 3.2).

Le template crée également le fichier gstnrvqm.h (voir Annexe B.2). Ici on retrouve entre autres la structure _GstNrvqm, qui permet de définir des variables et propriétés supplémentaires pour l'élément nrvqm.

On a besoin de différentes variables. Des variables qui contiennent des informations sur la dernière image qui est arrivée à l'élément nrvqm (lasttime, lasttimestamp, lastbuf). D'autres variables qui contiennent les informations sur les discontinuités, en tenant compte de la fenêtre d'analyse (quedisc).
Voici l'explication des variables ajoutées les plus importantes :

```
struct _GstNrvqm
{
  GstVideoFilter videofilter; //l'objet parent

  /* format */
  gint width; // largeur de l'image en pixels
  gint height; // hauteur de l'image en pixels
  ..

  /* properties */
  ..
  guint64 lasttime;     // Temps nrvqm lors de la dernière image
  guint64 lasttimestamp;// Temps d'affichage de la dernière image en ms
  GQueue * quedisc;     //pointeur sur la file des discontinuités
  GstBuffer * lastbuf;  //pointeur sur le buffer de la dernière image
}
```

La structure disc_t contient les informations concernant une discontinuité. Pour pouvoir décrire une discontinuité il faut en effet la durée de la discontinuité (duration), l'instant de démarrage de la discontinuité (mediatime). Mediatime permet également de calculer la durée des discontinuités qui ne sont pas encore terminées :

```
typedef struct disc_t {
    guint64    timestamp; // temps d'affichage de l'image du début en ms
    guint64    mediatime;     // temps nrvqm lors de l'image de début
    guint64    duration;      //durée de la discontinuité en ms
    float      f_dis_delta;   //delta calculé si applicable
} disc_t;
```

8.1.2 Implémentation du Modèle VQM adapté

Deux versions du modèle VQM basé sur la rupture de la fluidité existent. Dans la première version [PASTRANA2005], on ne calcule pas la variation de l'information temporelle à la fin de la discontinuité, le delta. Dans la deuxième version [PASTRANA2007], qui est présentée dans la section 4.3, les auteurs ont ajouté le calcul du delta pour améliorer davantage la précision de la métrique calculée.
Dans le cadre de ce mémoire on va se baser sur la version de [PASTRANA2005], sans le calcul du delta, vu que certains paramètres nécessaires au calcul du delta n'ont pas pu être déterminés.

On a déterminé que pour une utilisation dans le cas de la vidéo continue, on peut adapter le modèle VQM basé sur la rupture de la fluidité en utilisant une fenêtre d'analyse qui se déplace. On peut diviser en deux phases le modèle ainsi obtenu :

-une phase détection : qui comprend la détection des discontinuités, calculs des durées et sauvegarde de ces informations

-une phase de calcul : qui comprend le calcul de la métrique sur base des informations des discontinuités en tenant compte de la fenêtre d'analyse.

8.1.3 La phase détection

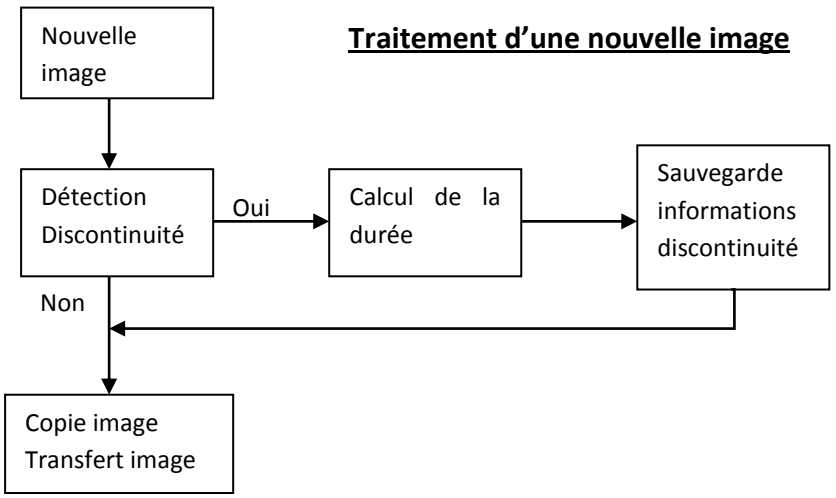


Figure 31 : schéma du traitement d'image lors de la phase détection.

La Figure 31 montre le schéma du traitement d'image lors de la phase détection.

Les grandes lignes de l'algorithme de traitement sont les suivantes :

```
Traitement nouvelle image() //gst_nrvqm_transform_ip(...)
{
Assignation variables ;

Détection des discontinuités ;
// gst_nrvqm_freeze_detect(…)

Adapter les informations des temps des variables temporaires ;
// temps d'affichages, nrvqm->lasttimestamp,…

Copie du buffer actuel (image + infos) vers buffer
temporaire ;
//copie outbuf vers  nrvqm->lastbuf

Envoi de l'image vers l'élément suivant ;
// se fait automatiquement
  }
```

La détection peut être implémentée de la façon suivante :
Dans [PASTRANA2005], une discontinuité est détectée, si la dérivée temporelle de la luminance de l'image est nulle : Detec=1⇔Y(x,y,t) – Y(x,y,t+1) = 0.
C'est-à-dire, on se base sur une comparaison pixel à pixel au niveau de la luminance (Y) de deux trames vidéo consécutives.
Les auteurs se basent sur l'hypothèse que, si la trame à décoder contient des erreurs et ne peut pas être décodée, beaucoup de décodeurs vont simplement transmettre la dernière trame sans erreur. Donc à l'affichage, on aura l'effet de gel d'image.

Dans le cas du framework GStreamer, cette hypothèse n'est pas satisfaite. En effet, une série de tests a montré que, lors de pertes de paquets réseau, le décodeur ne transmet pas la dernière trame qui n'avait pas d'erreur, mais n'envoie rien à l'élément suivant. C'est-à-dire tant qu'il y a des trames que le décodeur ne peut pas décoder, aucune donnée vidéo n'est transmise d'un élément vers les suivants dans le pipeline GStreamer. A l'affichage l'effet de gel d'image est bien présent, mais se base sur le fait que tant que la mémoire vidéo n'est pas changée, la même image est affichée. Donc la détection des discontinuités par la méthode présentée dans [PASTRANA2005] ne peut pas être implémentée dans notre cas.

Une possibilité pour réaliser la détection des discontinuités est d'utiliser le temps de passage de l'image dans l'élément NRVQM. Quand une image est traitée par NRVQM, on enregistre la valeur du temps (temps de l'horloge ou temps de lecture) à ce moment précis. Pour détecter une discontinuité, on peut comparer ce temps enregistré de la dernière image et le temps actuel pour l'image actuelle. Si la différence des deux temps est plus grande que le

seuil perceptuel (*threshold*) de détection d'une discontinuité, on aura une discontinuité avec une durée égale à la différence entre les deux temps. Les variables courtime (pour l'image actuelle) et nrvqm->lasttime (pour la dernière image) correspondent à ces deux temps.

Pour pouvoir déterminer l'histogramme lors du calcul de la métrique, on doit sauvegarder les informations concernant les discontinuités lors de la phase de détection. On choisit une structure de file (*queue*) pour sauvegarder ces informations, en effet, cette structure permet facilement d'ajouter une nouvelle discontinuité au début (*head*) de la file lors de la phase de détection et permet de supprimer facilement des discontinuités à la fin (*tail*) de la file lors de l'adaptation de la fenêtre d'analyse. La variable quedisc de type Gqueue correspond à cette file. Les informations sur la discontinuité sont enregistrées dans des structures de type disc_t.

Les grandes lignes de la fonction de détection sont les suivantes :

```
Détection des discontinuités ()  // gst_nrvqm_freeze_detect(…)
{
Assignation des variables ;

Si ((temps  image actuelle  - temps dernière image) > seuil
perceptuel)
     Alors
          {
               Créer une nouvelle instance de discontinuité ;
               Ajouter les informations de la discontinuité ;
               Ajouter cette instance au début de la file des
discontinuités ;
            }
  }
```

Il faut noter que cette manière de détecter les discontinuités peut poser un problème pour des flux vidéo utilisant de faibles fréquences d'affichages. La méthode de [PASTRANA2005] se base sur la comparaison des pixels de luminance de deux images consécutives et s'il y a correspondance, sur la différence de temps entre ces deux images pour détecter une discontinuité. La méthode proposée ici se base seulement sur la différence des temps entre ces deux images.

Le seuil de perception des discontinuités se situe entre 80 ms et 200 ms suivant [PASTRANA2005], une détection de 100% étant observée pour des discontinuités de 200 ms. Fixant le seuil par exemple à 100ms, si on se base maintenant sur un flux vidéo avec une fréquence d'affichage de 5 trames par seconde, les images sont espacées dans le temps de 1000/5 = 200 ms. Dans le cas normal sans pertes de paquets, la méthode de [PASTRANA2005] va travailler correctement et ne détectera pas des discontinuités, car deux images successives seront différentes. La méthode proposée ici se basera sur la différence

des temps entre les deux images (donc ici 200ms), qui sera plus grand que le seuil perceptuel (ici 100ms) et va détecter des discontinuités là où il n'y en a pas !
Ce problème entraîne deux conséquences :

-le seuil de perception devra être choisi assez grand par rapport à la fréquence d'affichage, en tenant compte qu'un seuil de perception de 200ms est détecté par 100% des personnes.

-la fréquence d'affichage ne doit pas diminuer en dessous d'une certaine valeur. En effet à 5 trames par seconde on arrive à un temps entre images de 200ms.

Un seuil de détection de 80ms nécessite une fréquence d'affichage plus grande que 12,5 FPS pour la méthode proposée ici. Les fréquences d'affichage dans le contexte étudié allant souvent de 5 à 15 FPS, cette valeur du seuil est trop faible. On choisit de se baser sur la valeur minimale de la fréquence d'affichage, donc 5 FPS, ce qui conduit à une valeur de seuil de perception de 200 ms.
Le choix de cette valeur aura naturellement un impact sur la précision de la métrique MOS calculée, comme on ne tient pas compte des discontinuités de moins de 200ms de durée.

Une possibilité pour amoindrir ce problème, qui ne sera pas approfondie ici, est d'utiliser des informations provenant d'autres éléments (éléments Gstreamer qui s'occupent de RTP/RTSP, élément décodeur,…) pour déterminer la fréquence d'affichage du flux vidéo et d'adapter automatiquement le seuil de perception en conséquence, pour augmenter la précision. Il faut alors tenir compte des éventuelles adaptations automatiques de la qualité vidéo par les serveurs de streaming, comme le frame dropping, stream thining, etc., qui diminuent la fréquence d'affichage (voir Section 6.2.2).

Un avantage de la méthode de détection des discontinuités choisie est que la puissance de calcul nécessaire est très faible. En effet la détection se base principalement sur une simple comparaison entre deux temps et on n'utilse pas d'opération nécessitant une puissance de calcul élevée.

8.1.4 La phase de calcul

Calcul de la métrique MOS

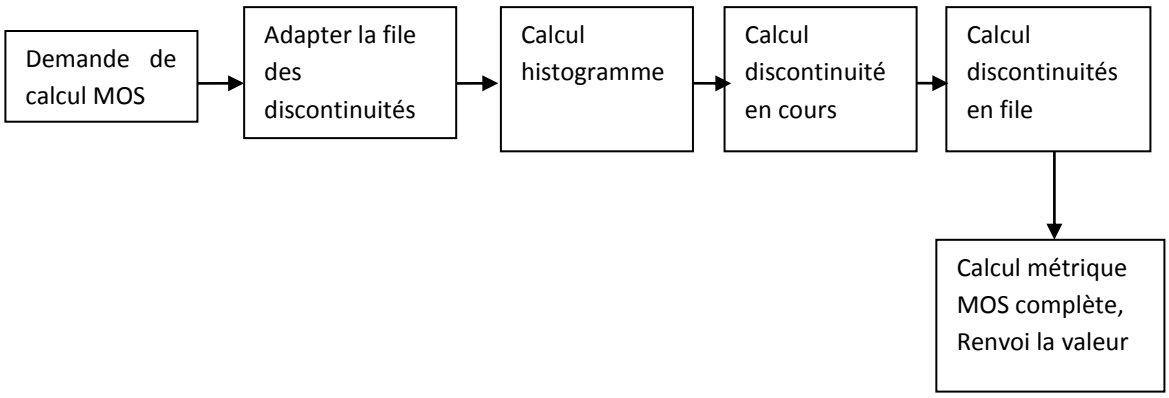


Figure 32: Schéma du calcul de la métrique MOS

Le modèle de prédiction de [PASTRANA2005], qui ne tient pas compte de la variation de l'information temporelle, est le suivant :

Métrique MOS = $mos_{ref} - d_{total,}$

d_{total} = min $\{d_{pooling}, d_{max}\}$,

$d_{pooling}$ = $[\sum_{tg=Tmin}^{Tmax} d_{tg}]^{1/2}$,

d_{tg} = $n(t_g) \times [\hat{e}(t_g)]^{p(n(tg))}$,

$\hat{e}(t_g)$ = $mos_{ref} - q(t_g)$,

$q(t_g)$ = $m_{max} - ((m_{max} - m_{min}) / (1 + (b / t_g)^s))$,

$p(n(t_g))$ = $p_{max} - ((p_{max} - p_{min}) / (1 + (c / n(t_g))^r))$.

Les grandes lignes de l'algorithme sont les suivantes :

```
Calcul métrique MOS() // gst_nrvqm_calc_vqm (...)
{
Assignation variables ;

Pour chaque élément de la file des discontinuités (quedisc):
      Si le temps de terminaison de la discontinuité se trouve dans la
fenêtre d'analyse des discontinuités, alors mettre à jour l'histogramme des
discontinuités (calc_ntg (…)).
      Sinon, supprimer cette discontinuité de la file ;


Déterminer si une discontinuité est en cours :
      Si le temps depuis la dernière image (par rapport au temps actuel)
est plus grand que le seuil perceptuel, alors mettre à jour l'histogramme
(calc_ntg(…)) et calculer la partie dégradation de qualité due à cette
discontinuité (f_dpooltmp) ;

Calculer la partie dégradation de qualité due aux discontinuités dans la
file (f_dpooltmp);
```

```
Calculer la dégradation sur la fenêtre d'analyse ; (f_dpooling=racine
carrée de f_dpooltmp)

Calculer la dégradation totale, qui est égale au minimum entre la
dégradation sur la fenêtre d'analyse (f_dpooling) et la valeur dmax
(dégradation maximale fixée) ;

Calculer la valeur du MOS finale (avec une valeur minimale de 10), et
renvoyer la valeur ;

  }
```

Le calcul de la métrique peut être implémenté de la façon suivante :
Pour déterminer si le temps de terminaison de la discontinuité se trouve dans la fenêtre d'analyse, on vérifie si ((temps de début de la discontinuité + sa durée) – temps actuel) < durée de la fenêtre d'analyse (=10 secondes).

L'histogramme des discontinuités représente la répartition des durées des discontinuités apparues pendant la fenêtre d'analyse. Pour pouvoir calculer l'histogramme il faut d'abord déterminer le nombre des classes et l'intervalle des classes pour représenter les données.
Une idée est de se baser sur la dégradation de la qualité vidéo causée par une seule discontinuité d'une certaine durée pour spécifier les classes. Cette dégradation d'une certaine durée correspondra à une certaine valeur MOS. Le nombre de classes se base sur le nombre de valeurs pour l'échelle utilisée dans le MOS (voir Section 3.1), c'est-à-dire 5 classes. On va déterminer les intervalles entre les classes de telle façon que toutes les dégradations qui correspondent à une même valeur de l'échelle MOS (très gênant, gênant, peu gênant, perceptible, imperceptible) correspondent à une même classe pour représenter les données.
Dans [PASTRANA2004], les auteurs présentent les résultats des mesures sur la qualité perçue en fonction de la durée d'une discontinuité moyennée sur un ensemble de contenus vidéo variés (Figure 33).

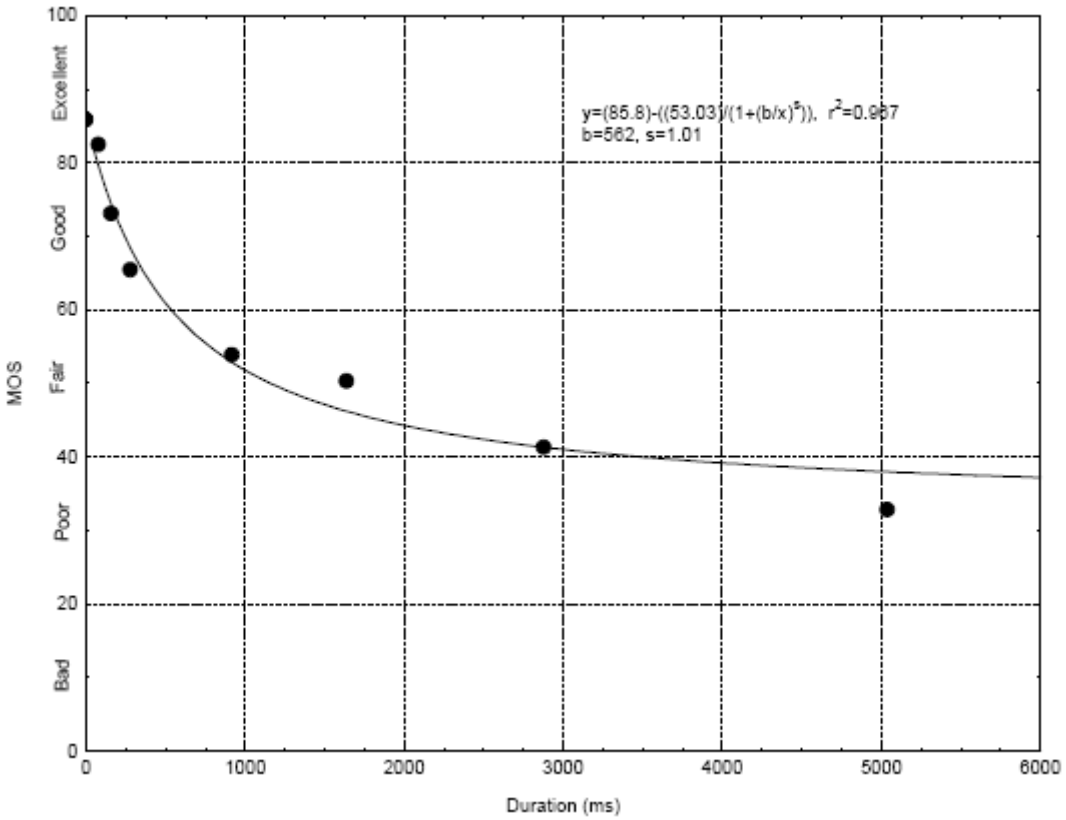


Figure 33: Qualité perçue en fonction de la durée d'une discontinuité moyennée sur un ensemble de contenus vidéo variés [PASTRANA2004].

Ces résultats nous permettent de calculer les intervalles entre les classes par la fonction :

$y=(85.8)-((53.03)/(1+(b/x)^{s}))$

avec b=562 et s=1.01

Le Tableau 15 montre les intervalles des classes pour l'histogramme ainsi obtenu.

Dégradation	imperceptible	perceptible	peu gênant	gênant	très gênant
Durée de la dégradation	de 0 à 70,46 ms	de 70,46 à 532 ms	de 532 à 3495 ms	au dessus de 3495 ms	pas déterminée

Tableau 15: Intervalles des classes de l'histogramme.

Les valeurs des paramètres du modèle sont les suivants [PASTRANA2005] :

mos_{ref} (Q_{ref}) = 95

d_{max} = 90

T_{min} > seuil de détection donc dans notre cas > 200 ms

T_{max} = temps analyse globale donc dans notre cas est égale à la durée de la fenêtre d'analyse donc 10s

Les paramètres suivants peuvent être déterminés à partir de la Figure 33:

m_{max} = 85,8

m_{min} = 32,77

b = 562

s = 1,01

Les paramètres suivants peuvent être déterminés à partir de la distribution des puissances optimales [PASTRANA 2005] de la Figure 34:

pmax = 2,017

pmin = 1,1131

c = 27

r = 1,5

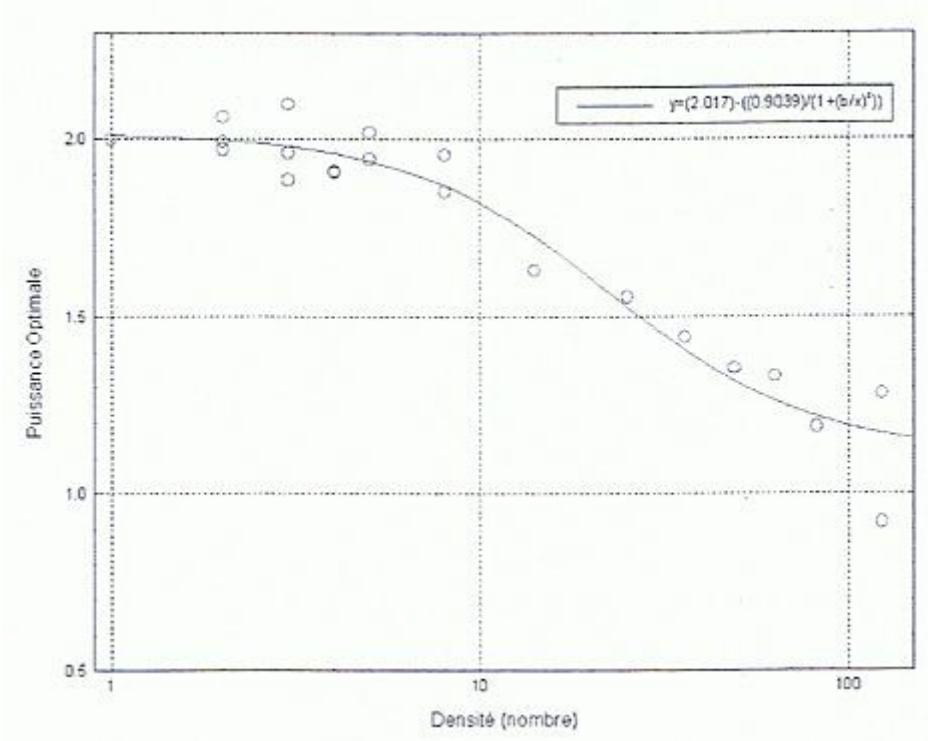


Figure 34: Distribution des puissances optimales [PASTRANA2005].

Le fait d'utiliser des intervalles des classes pour l'histogramme, requiert une adaptation du calcul de $p(n(t_g))$. En effet $p(n(t_g))$ doit tenir compte ici des toutes les discontinuités d'un intervalle d'une classe et pas seulement des discontinuités d'une même durée.

Donc on va renommer $n(t_g)$ en $n(t_{gc})$ pour $p(n(t_g))$ afin de le différencier du nombre de discontinuités d'une même durée. L'équation devient :

$$p(n(t_{gc})) = p_{max} - ((p_{max} - p_{min}) / (1 + (c / n(t_{gc}))^r))$$

Le modèle de prédiction de [PASTRANA2005] adapté devient :

Métrique MOS = $mos_{ref} - d_{total}$,

$d_{total} = \min \{d_{pooling}, d_{max}\}$,

$d_{pooling} = [\sum_{tg=Tmin}^{Tmax} d_{tg}]^{1/2}$,

$d_{tg} = n(t_g) \times [\hat{e}(t_g)]^{p(n(tgc))}$,

$\hat{e}(t_g) = mos_{ref} - q(t_g)$,

$q(t_g) = m_{max} - ((m_{max} - m_{min}) / (1 + (b / t_g)^s))$,

$p(n(t_{gc})) = p_{max} - ((p_{max} - p_{min}) / (1 + (c / n(t_{gc}))^r))$.

Dans [PASTRANA2005] les auteurs ont déterminés une valeur MOS minimale. Cette valeur est égale à 10.
Donc,
Métrique MOS =max { ($mos_{ref} - d_{total}$), 10}

On représente ici l'implémentation pour le calcul de la partie dégradation de qualité due aux discontinuités dans la file (f_dpooltmp);

```
{
Pour chaque discontinuité dans la file
      {
      Déterminer n(tgc);    // i_ntg =  i_tab_ntg[calc_ntg(tmpdisc->duration)
      Calculer p(n(tgc));
      Calculer ê(tg);      //f_etgi= mosref  -  mmax-((mmax - mmin)/(1 +(b/tg)s))

      f_dpool_tmp = f_dpool_tmp + ê(tg) ^p(n(tgc)) ;
      }
}
```

La puissance de calcul pour le calcul de la métrique VQM est très faible. Principalement on calcule la dégradation partielle engendrée par chaque discontinuité dans la file des discontinuités, $[\hat{e}(t_g)]^{p(n(tgc))}$. Pour chaque discontinuité on a 8 additions/soustractions, 4 divisions, 3 exposants à calculer ce qui nécessite seulement une faible puissance de calcul.

8.1.5 Implémentation de la signalisation de la métrique MOS

L'élément GStreamer NRVQM doit signaler à l'élément GStreamer GstRTSPSrc la métrique MOS calculée. Le framework GStreamer possède un bus qui permet la communication entre les éléments via les évènements. On va donc implémenter une fonction qui crée un nouvel évènement spécifique à la métrique MOS et envoie cet évènement sur les bus GStreamer vers l'amont du pipeline.

Les grandes lignes de l'algorithme sont les suivantes :

```
Envoi métrique MOS() // gst_sendmosv (…)
{
Assignation variables ;

Appel de la fonction de calcul de la métrique MOS ;
// gst_nrvqm_calc_vqm (...)

Créer un évènement (upstream) personalisé à base d'une structure
GStreamer, qui a comme nom de la structure = 'video/vqm', nom de
champ = 'vqmmos ' et la valeur de la métrique calculée ;

Envoyer cet évènement sur le bus GStreamer ;
}
```

Une problématique qui se pose pour la signalisation de la métrique MOS de l'élément NRVQM vers l'élément qui s'occupe du RTP/RTSP, se situe au niveau de la synchronisation entre le moment de l'envoi de la métrique MOS de l'élément NRVQM et le moment que l'élément qui s'occupe du RTP/RSTP a besoin de la métrique pour pouvoir la transmettre via un paquet RTCP XR QoE.

En effet, comme on l'a vu, l'élément RTP envoie périodiquement les paquets RTCP vers le serveur de streaming. Cette fréquence (variable) d'envoi est déterminée suivant le [RFC3550]. L'idéal serait que l'élément qui s'occupe du RTP/RTSP envoie une 'demande de calcul ' de la métrique sur le bus GStreamer. Cette demande est interceptée par l'élément NRVQM, qui calcule la métrique MOS et la renvoie via un événement sur le bus GStreamer. L'élément qui s'occupe du RTP/RTSP reçoit cet évènement, ajoute un paquet RTCP XR QoE au paquet RTCP et transmet le tout vers le serveur.
La problématique se situe au niveau du bus GStreamer. Les évènements qui sont envoyés sur le bus sont des évènements à sens unique, asynchrones ('*One-Way*'), c'est-à-dire si un élément envoie un évènement sur le bus, il n'attend pas de réponse d'un autre élément.
Dans notre cas, l'élément RTP/RTSP ne va donc pas attendre la réponse de l'élément NRVQM, et il y aura un risque que l'évènement envoyé par NRVQM arrive après la transmission du paquet RTCP.
Une possibilité serait que l'élément RTP attende la réponse de l'élément NRVQM. Ceci pose néanmoins deux problèmes :

- Comme l'élément doit aussi s'occuper de toute la communication RTP, ce temps d'attente risque de perturber son fonctionnement.
- Il n'y a aucune assurance qu'il y ait un élément de type NRVQM dans le pipeline, donc il n'y a pas de garantie qu'il y ait une réponse.

C'est pourquoi on a choisi qu'il n'y ait pas de demande MOS de la part de l'élément qui s'occupe du RTP/RTSP, mais que l'élément NRVQM envoie périodiquement un évènement avec la métrique MOS calculée sur le bus GStreamer. Cette technique entraîne naturellement une perte de précision de la métrique envoyée par l'élément qui s'occupe du RTP/RTSP, comme la métrique envoyée par l'élément RTP/RTSP n'a pas été calculée au moment de l'envoi du paquet RTCP, mais un certain temps avant.
Pour limiter cette perte de précision, il faut déterminer une période de signalisation adaptée. On a fixé le seuil de perception à 200ms. Donc il ne sert à rien de fixer la période de signalisation en dessous de 200ms. D'un autre côté, dans notre cas d'étude, la durée entre deux paquets RTCP est de plusieurs secondes et il ne s'agit pas d'effectuer un feed-back en temps réel vers le serveur de streaming. Donc on choisit une période de 2* seuil de perception, c'est-à-dire 400 ms pour la période de signalisation.
On utilise la fonction gst_nrvqm_start (…) qui est appelée automatiquement quand l'élément NRVQM commence le traitement, pour définir un timer qui appelle tous les 400 ms la fonction d'envoi de la métrique MOS() `(gst_sendmosv (…))`.

8.2 Adaptation des plugins RTP/RTSP

Les éléments des plugins qui s'occupent du RTP/RTSP sont utilisés dans GStreamer pour transmettre et contrôler le flux RTP. Ces éléments doivent être adaptés pour intercepter l'évènement envoyé par l'élément NRVQM, qui contient la valeur de la métrique MOS calculée, et transmettre cette métrique vers le serveur via un paquet RTCP XR QoE [CLARK et al, 2008].
On va d'abord présenter les différents éléments RTP/RTSP principaux, nécessaires pour réaliser les adaptations, puis on va proposer une implémentation des adaptations nécessaires.

8.2.1 Présentation des éléments RTP/RTSP

L'élément GStreamer GstRTSPSrc s'occupe de la communication RTSP entre le client média et le serveur. Il est compatible avec le standard [RFC2326]. Il va instancier en interne un gestionnaire de sessions RTP, l'élément GStreamer GstRtpBin, qui s'occupe entre autres du traitement des paquets RTCP, de la remise en ordre de paquets RTP et fournit l'horloge pour le pipeline.
GstRtpBin peut gérer plusieurs sessions RTP, chaque session étant représentée par un élément GStreamer GstRtpSession. La session peut être utilisée pour envoyer et recevoir des paquets RTP et RTCP en suivant le standard [RFC3550]. GstRtpSession se base sur un module interne RtpSession, qui n'est pas un élément GSteamer, pour réaliser ces fonctionnalités. C'est le module qui est responsable de la construction et de l'envoi des paquets RTCP.
GstRtpBin regroupe d'autres fonctionnalités comme le multiplexage/démultiplexage, le contrôle de jitter, … qu'on ne va pas utiliser dans notre cas.
La Figure 35 montre les éléments nécessaires pour notre cas.

Figure 35: Les éléments GStreamer utilisés pour l'implémentation du feed-back MOS.

Le seul élément GSteamer qui soit 'visible' dans le pipeline GStreamer est l'élément GstRTSPSrc. En effet, cet élément va instancier en interne le gestionnaire de session GstRtpBin. Ceci a comme conséquence que les évènements envoyés par NRVQM via le bus GStreamer peuvent seulement être interceptés par l'élément GstRTSPSrc et pas par les éléments GstRtpBin ou GstRtpSession.

8.2.2 Implémentation du feed-back RTCP

On a choisi de renvoyer la métrique MOS en utilisant le paquet RTCP XR QoE de [CLARK et al, 2008]. Ce paquet est ajouté aux paquets RTCP RR pour former un paquet composé. Ce paquet composé est envoyé au serveur suivant une fréquence d'envoi déterminée par le standard [RFC3550].
Pour pouvoir implémenter cette fonctionnalité, on utilise le module RtpSession. Celui-ci gère les paquets RTCP et dispose d'une fonction, `rtp_session_on_timeout`(…) (voir Annexe C.8), qui est appelée périodiquement pour effectuer des tâches de maintenance, entre autres pour ajouter et envoyer les paquets RTCP RR.
On adapte cette fonction telle que, si des paquets RTCP existent déjà dans le paquet composé, alors on ajoute à la fin le paquet RTCP XR QoE avec la dernière valeur de la métrique MOS envoyée par l'élément NRVQM. La métrique MOS envoyée par l'élément NRVQM est stockée dans la variable de l'objet RtpSession : i_mosv (voir Annexe C.6).
Si on n'a pas encore reçu un évènement de la part de l'élément NRVQM avec la métrique MOS, i_mosv est égale à 0 et le paquet RTCP XR QoE ne sera pas envoyé.

Les principales étapes de la fonction adaptée sont les suivantes :

```
rtp_session_on_timeout (…) //fonction adaptée
{
Assignation des variables ;

Tâches de maintenance ;
//(calcul nouvel intervalle, nettoyages,…) inchangées

Si (temps actuel = temps envoi paquet RTCP), // is_rtcp_time(…)
     Alors créer un paquet RTCP RR et ajouter les blocks de rapport
RTCP RR pour chaque source ;
//fonctions inchangées

S'il existe un paquet  RTCP RR,
     Alors Si i_mosv >0,   //donc on a reçu un évènement de NRVQM
Alors Ajouter un paquet et un block de rapport RTCP XR QoE      ;

// gst_rtcp_buffer_add_packet(…), gst_rtcp_packet_add_xr(…)


Autres tâches de maintenance ; //fonctions inchangées

Envoi du paquet RTCP Composé ;  //fonctions inchangées

}
```

La fonction `gst_rtcp_buffer_add_packet(…)` crée un nouveau paquet de type GST_RTCP_TYPE_XR, c'est-à-dire un paquet RTCP de type 207 (voir Annexe C.9). C'est une fonction existante.

La fonction gst_rtcp_packet_add_xr (&rtcpapp,sess->source->ssrc, 1297044310,sess->i_mosv), voir Annexe C.7, ajoute le bloc de rapport RTCP XR QoE de [CLARK et al, 2008] au paquet RTCP. Cette fonction a été ajoutée dans RtpSession.
La Figure 36 montre la structure du bloc de rapport, voir la Section 5.1.4.

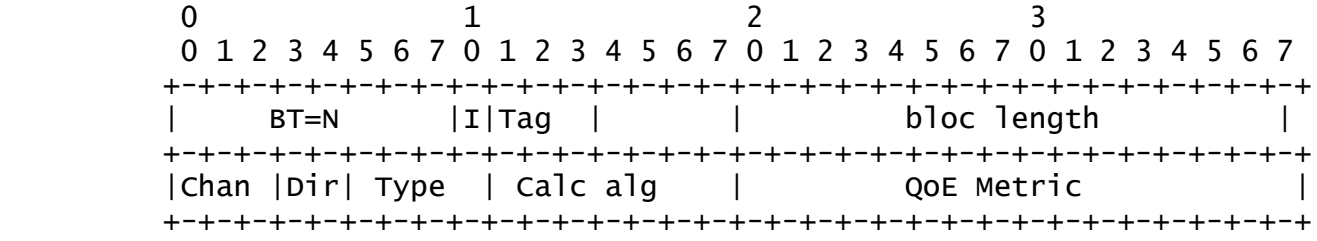


Figure 36: Structure du bloc de rapport QoE [CLARK et al].

Les valeurs des champs sont les suivants :
BT = N'est pas encore standardisé, on prend la valeur décimale 100, qui n'est pas encore utilisée.
I = comme on calcule la métrique MOS indépendamment des intervalles des paquets RTCP, ce champ n'a pas de signification dans notre cas. On prend la valeur binaire '0'.
Tag = Dans le cadre de ce mémoire, on se situe dans un contexte de communication RTP unicast avec un seul serveur source. Dans ce cas, ce champ n'est pas absolument nécessaire. On prend la valeur binaire '000'.
Chan = Dans notre contexte on ne dispose qu'un seul canal, donc on prend la valeur binaire '000'.
Dir = sens de la communication, la valeur n'est pas définie dans [CLARK et al, 2008]. On prend la valeur binaire '00'.
Type = On prend la valeur binaire '0100' qui correspond à *Video Quality MOS*.
Calc alg = L'algorithme VQM implémenté dans notre contexte, n'est pas encore défini. On choisit la valeur décimale 4, qui n'est pas encore prise.
QoE *Metric* = valeur de la métrique MOS calculée en type "*integer scaled representation* 8:8"

Le *Tag* correspond à un identifiant qui est associé à un bloc d'identification de mesure qui est défini dans le draft [HUNT et al, 2009]. Ce bloc d'identifiant comprend des informations sur les serveurs sources (SSRC, intervalle de rapport,…). Lorsqu'on utilise plusieurs types de blocs de métriques RTCP XR différents, l'utilisation d'un bloc d'identification de mesure évite d'ajouter des informations redondantes dans ces blocs.
Dans le cadre de ce mémoire, on se situe dans un contexte de communication RTP unicast avec un seul serveur source. En plus la métrique MOS calculée est indépendante de

l'intervalle de rapport. Comme ce bloc d'identification n'est absolument nécessaire dans notre contexte, on a choisi de ne pas l'implémenter pour limiter l'ampleur de ce mémoire.

L'implémentation du type "*integer scaled* 8:8" se fait de la manière suivante :

-Le premier octet du champ comprend la partie entière de la métrique MOS

-Le deuxième octet du champ correspond à la parie décimale de la métrique MOS en calculant : 256 X partie décimale de la métrique MOS et en arrondissant le résultat vers le bas. Par exemple 34,4789 est représenté par 34 pour le premier octet et 256 X 0,4789 = 122,5984 donc 122 pour le deuxième octet.

8.2.3 Traitement de l'évènement 'video/vqm'

On a vu que l'élément NRVQM envoie périodiquement un évènement qui contient la métrique MOS sur le bus GStreamer. L'élément GstRTSPSrc va intercepter cet évènement. L'élément GstRTPSession envoie le paquet RTCP XR QoE vers le serveur de streaming via son module interne RtpSession. Le module RtpSession représente la métrique MOS par la variable i_mosv.

On propose ici une possibilité pour intercepter l'évènement 'video/vqm' par l'élément GstRTSPSrc et de faire parvenir la valeur de la métrique vers la variable i_mosv du module RtpSession.

L'élément GstRTSPSrc dispose d'une fonction qui traite les évènements du bus GStreamer sur le pad 'source', c'est-à-dire les évènements envoyés vers l'amont du pipeline. La fonction s'appelle gst_rtspsrc_handle_src_event(…)(Voir Annexe C.1). On va adapter cette fonction pour prendre en compte les évènements 'video/vqm' qui contiennent un champ 'vqmmos'.

Les étapes principales de la fonction adaptée sont :

```
Traitement évènements source( ...) ; // gst_rtspsrc_handle_src_event(…)
{
Assignations variables ;

Switch(type de l'évènement)
      {
       Si (type de l'évènement == évènement personnalisé (upstream)
            ET (évènement contient la métrique 'vqmmos')
            Alors Faire passer la métrique ; // g_object_set(…)

        Traitement autres évènements ; //inchangés
      }

Transmettre l'évènement vers l'élément suivant si nécessaire ; //inchangé
}
```

On a vu que tous les éléments GStreamer sont dérivés de la classe GObject et on peut donc les considérer comme des objets GObject. L'élément GstRTSPSrc n'a donc pas un accès direct sur la variable i_mosv du module RtpSession.
Pour faire passer la métrique MOS de l'élément GstRTSPSrc vers la variable i_mosv du module RtpSession, on va utiliser les propriétés des GObjects. Pour les classes GObject, il est possible d'ajouter des propriétés à la classe, via la fonction g_object_class_install_property (…), et de mettre à jour cette propriété via la fonction g_object_set_property(…). Lorsqu'un objet reçoit une demande de mise à jour d'une propriété, il appelle une fonction (_set_property(…)), qui permet de traiter la mise à jour de la propriété.

Dans notre cas, l'élément GStreamer GstRTSPSrc dispose d'une référence sur le gestionnaire des sessions RTP, GstRtpBin. Celui-ci dispose d'une référence sur l'élément GstRtpSession qui gère une session RTP. GstRtpSession dispose d'un accès direct sur la variable i_mosv du module RtpSession.
L'idée est d'ajouter une propriété 'Prop_imosv' ou 'imosv' à l'élément GstRtpBin et à GstRtpSession. Lors d'une mise à jour de la propriété PROP_IMOSV de l'élément GstRtpBin avec la métrique MOS venant de GstRTSPSrc, celui-ci va mettre à jour la propriété de GstRtpSession, qui va assigner la variable i_mosv du module RtpSession.

Les étapes principales de l'implémentation dans GstRtpBin sont :

```
Ajout et définition de la propriété à la classe dans la fonction ;
gst_rtp_bin_class_init(…)
{
 …
//ajouter et définir la propriété 'PROP_IMOSV', 'imosv'
g_object_class_install_property (gobject_class, PROP_IMOSV,
      g_param_spec_float ("imosv", "IMOSV","MOS VIDEO", 1,
          G_MAXFLOAT, DEFAULT_IMOSV, G_PARAM_READWRITE));
…
}

Demande de mise à jour de la propriété imosv de GstRtpBin ;
gst_rtp_bin_set_property(…) // Voir Annexe C.2
{
…
S'il y a une demande de mise à jour de la propriété PROP_IMOSV,
     Alors faire propager la métrique vers GstRtpSession ;
//    case PROP_IMOSV:
//    gst_rtp_bin_propagate_property_to_rtpsession (rtpbin, "imosv",
value);

…
```

```
}

Propager la propriété à GstRtpSession ;
gst_rtp_bin_propagate_property_to_rtpsession(…)//Voir Annexe C.3
{
…
Pour toutes les sessions GstRtpSession, mettre à jour la propriété
imosv de la session GstRtpSession ;
// g_object_set_property (G_OBJECT (sess), name, value);
…
}
```

Les étapes principales de l'implémentation dans GstRtpSession sont :

```
Ajout et définition de la propriété à la classe dans la fonction ;
gst_rtp_session_class_init(…)
{
 …
//ajouter et definer la propriété 'PROP_IMOSV', 'imosv'
g_object_class_install_property (gobject_class, PROP_IMOSV,
      g_param_spec_float("imosv", "IMOSV","MOS VIDEO", 0,
          G_MAXFLOAT, DEFAULT_IMOSV, G_PARAM_READWRITE));
…
}

Demande de mise à jour de la propriété imosv de GstRtpBin ;
gst_rtp_session_set_property(…)//Voir Annexe C.4
{
…
S'il y a une demande de mise à jour de la propriété PROP_IMOSV,
     Alors assigner i_mosv avec la valeur de la métrique ;
//   case PROP_IMOSV:
//   rtp_session_set_imosv(priv->session, g_value_get_float(value));

…
}


Assigner i_mosv avec la valeur de la métrique ;
gst_rtp_session_set_mosv (…)//Voir Annexe C.5
{
  …
   Assigner i_mosv avec la valeur MOS ;
   //session->priv->session->i_mosv=mosv;
 …
}
```

9 Analyse des résultats

L'objectif de ce chapitre est de décrire l'environnement utilisé pour tester la solution proposée et d'analyser les résultats de ces essais. Comme le mémoire se limite à la partie client d'un système d'adaptation de la qualité vidéo, les essais et l'analyse des résultats vont seulement prendre en compte la partie client et se limiterons à la vérification de la valeur MOS calculée et le feed-back de cette valeur vers le serveur de streaming.

9.1.1 L'environnement de test

Pour pouvoir tester la solution proposée, on a besoin d'un serveur de streaming qui soit compatible avec les protocoles RTP/RTSP de base et supporte les flux encodés en H.264, d'un client média où la solution va être déployée et d'un émulateur de WAN qui permet de simuler une connexion WAN.

Pour le serveur de streaming, on a choisi le programme à code ouvert (Open source) Darwin de APPLE. Darwin est compatible avec les protocoles RTP/RTSP de base. Il permet également d'envoyer des flux média encodés avec le CODEC H.264. Il est installé sur un ordinateur avec le système d'exploitation Ubuntu 8.10 Intrepid Ibex.

Le client média est composé d'un ordinateur avec le système d'exploitation Ubuntu 9.10 sur lequel on a installé le plugin GStreamer NRVQM. La configuration requise et l'installation du plugin est décrite dans l'Annexe D.

L'émulateur de WAN est composé d'un ordinateur avec le système d'exploitation LINUX (KNOPPIX) utilisant le programme d'émulation de WAN WANem (*Wan Area Network Emulation*, http://openmaniak.com/wanem_network.php). On démarre l'ordinateur avec un CD qui comporte le système d'exploitation et le programme WANem (LiveCD). WANem permet d'émuler différentes qualités de liaison réseau. On peut ainsi configurer une bande passante spécifique, simuler une perte de paquet, simuler des erreurs dans les paquets, …
L'ordinateur WANem se trouve entre le client et le serveur pour pouvoir adapter les paquets réseau. Dans le cas d'une utilisation de WANem dans un réseau local où les différents ordinateurs se trouvent dans le même sous-réseau, il faut créer un routage. La Figure 37 montre le schéma de l'environnement de test. On a crée une route entre le serveur de streaming et WANem et entre le Client média et WANem. Ainsi les paquets transmis entre le serveur et le client passent toujours par WANem.

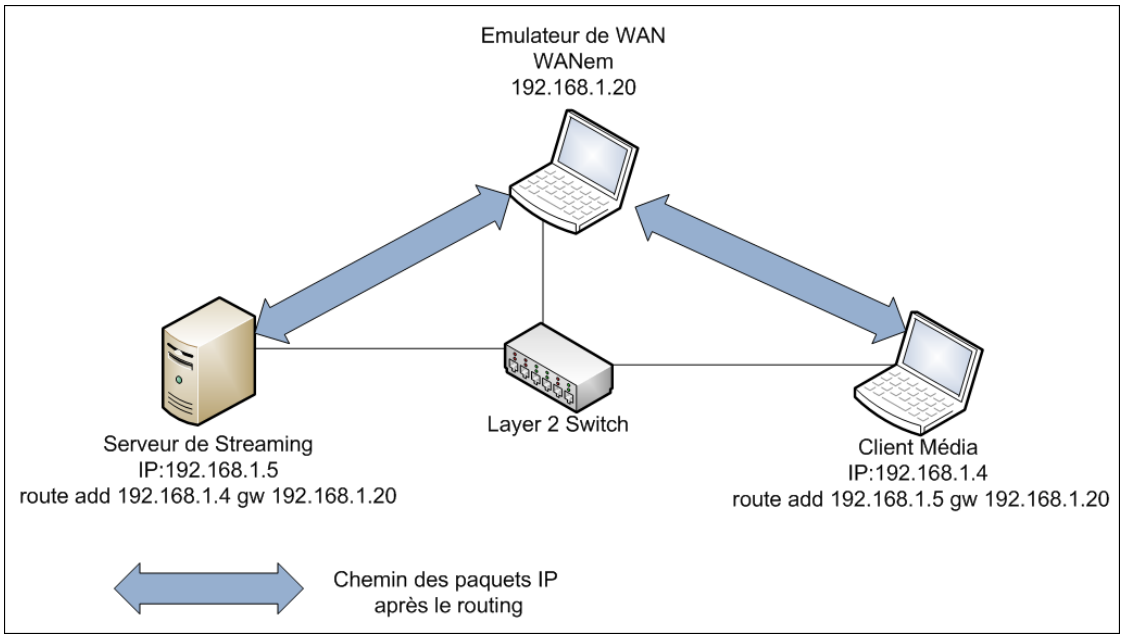


Figure 37: Schéma de l'environnement de test.

Pour effectuer les tests de streaming, on a choisi d'utiliser un fichier média dont la partie vidéo a une résolution de 352X288, est encodée en H.264, 15 FPS, 3053 trames en tout. La partie Audio est en stéréo, 32 kHz, encodé en MPEG AAC (mp4a). La durée de la séquence vidéo est de 102 secondes. Les données audio et vidéo correspondent à 8319 kB. Donc on a un débit moyen de 8319 X 8 / 102 = 652 Kbits/seconde.
L'utilisation d'une autre vidéo avec une résolution de 320X240, 15 FPS et encodée avec 320 Kbit/s provoquait continuellement des pertes de référence vers la configuration PS (Program Stream) lors de la présence des perturbations (et même après). Ce problème n'est pas liée aux plugins développées/adaptées pour ce mémoire. L'effet des perturbations sur certains éléments GStreamer semble tel, que même après la fin des perturbations, ces éléments ne fonctionnement plus correctement, lors de l'utilisation de fichiers vidéo encodés avec un faible débit binaire.
Avec la vidéo encodée à 652 Kbit/s on n'a pas eu de problème de perte de référence PS et le framework GStreaemer fonctionne comme prévu.

On utilise l'application GStreamer gst-launch pour construire/contrôler le pipeline GStreamer. On utilise la commande suivante pour jouer la vidéo test.mp4 du serveur de streaming et d'afficher la métrique MOS renvoyée dans la fenêtre du terminal :

gst-launch --gst-debug=nrvqm_mosv:4,rtpsession_rtcp:5 playbin uri=rtsp://192.168.1.5/test.mp4 video-sink="nrvqm ! xvimagesink"

9.1.2 Les essais effectués

L'élément NRVQM réalise principalement deux fonctions, le calcul de la métrique MOS sur base des discontinuités de fluidité [PASTRANA2005] et l'envoi de cette métrique MOS au serveur via un paquet RTCP XR QoE [Clark et al, 2008]. Les essais proposés ici devront tester ces fonctionnalités.

Pour pouvoir compares les différents essais, ceux-ci sont effectués de la manière suivante :

- pour les 10 premières secondes, la communication entre le serveur et le client n'est pas perturbée.

-de la 10ème à la 40ème seconde on utilise WANem pour perturber la communication entre le serveur et le client. (Bande passante limitée, perte de paquets)

-à partir de la 40ème seconde on supprime la perturbation engendrée par WANem.

-on arrête la lecture à la 80ème seconde.

Pour le calcul de la métrique MOS :

On a vu que dans le contexte de la vidéo mobile, la bande passante peut varier, en plus on peut avoir des pertes de paquet.

Donc on effectue deux séries de test :

-une série de test avec des bandes passantes différentes

-une série de test avec des taux de pertes de paquet différents

Pour la série de test avec des bandes passantes différentes, on dispose d'un flux média qui a besoin d'une bande passante moyenne de 652kbit/s pour transmettre l'audio et la vidéo sans compter les en-têtes des protocoles RTP/UDP/IP. On va choisir 5 bandes passantes de telle façon que la bande de passante la plus grande n'engendre pas de dégradation et la bande passante la plus petite engendre une dégradation importante. Les bandes passantes choisies sont : 1000Kbit/s, 800Kbits/s, 600Kbits/s, 500Kbits/s, 400 Kbits/s.

Les auteurs de [CERMAK2005] ont effectué des tests concernant l'effet de la perte de paquets sur la qualité vidéo. Ils ont utilisés des taux de pertes de paquets allant de 0% à 4% et ont constaté qu'à 4% la qualité se dégrade fortement. Pour la série de test avec des taux de pertes de paquets, on choisit donc les taux suivants : 0,1%, 0,5 %, 1%, 2%, 5%. En considérant qu'en dessous de 0,1% les dégradations sont minimales et à 5% les dégradations sont très importantes.

Pour le feed-back de la métrique MOS via RTXP XR QoE :

On effectue un essai avec une perte de paquets de 1%. On va comparer comment la métrique MOS calculée se comporte par rapport a la métrique envoyée au serveur.

9.1.3 Analyse des résultats

On va présenter un extrait des résultats des essais effectués.

Pour tous les graphiques présentés ici, l'ordonnée correspond à la valeur de la métrique MOS qui peut varier de 0 à 100. L'abscisse correspond au temps en ms.

Analyse des résultats des essais sur le calcul de la métrique MOS :

On a comparé les valeurs calculées par le plugin NRVQM avec les valeurs calculées manuellement pour vérifier l'exactitude du calcul.
Pour l'essai considéré, on a une première discontinuité vers 14186 ms qui dure 236 ms. La valeur MOS était à 95 avant la discontinuité, après le MOS calculé est descendu à 69,787. Une deuxième discontinuité s'est présentée vers 18002, qui a une durée de 240 ms. La valeur du MOS calculé après la discontinuité est de 59,861.
Calculons manuellement la valeur MOS après la première discontinuité :
$p(n(t_{gc}))$ se base sur une seule discontinuité dans l'intervalle de classe, donc $n(t_{gc})=1$.
$p(n(t_{gc}))$ = 2.017 – ((2.017 – 1.1131) / (1+(27/1))^1.5)) = 2.010899
q(236) = 85.8 – ((85.8-32.77) / (1+(562/236)^1.01)) = 70,212606023
et_g = 95 – 70,212606023= 24,787393977
$d_{pooling}$ = [(24,787393977)$^{\wedge 2,010899}$]$^{\wedge 1/2}$ =25,224858036
MOS = 95 – 25,224858036= 69,775141964 => ce qui correspond à la valeur calculée par NRVQM (69,787) en tenant compte des arrondis.

Calculons manuellement la valeur MOS après la deuxième discontinuité :
A ce moment, les deux discontinuités se trouvent dans la fenêtre d'analyse de 10s. En plus, les deux discontinuités appartiennent à un même intervalle de classe (perceptible) donc $n(t_{gc})=2$.
p(n(tgc)) = 2.017 – ((2.017 – 1.1131) / (1+(27/2))^1.5)) = 2,000629
q(240) = 85.8 – ((85.8-32.77) / (1+(562/240)^1.01)) = 70,120981618
$et_{g(240)}$ = 95 – 70,120981618= 24,879018382
$d_{pooling}$ = [(24,787393977)$^{\wedge 2,000629}$+ (24,879018382)$^{\wedge 2,000629}$]$^{\wedge 1/2}$ =35,15501295
MOS = 95 – 35,15501295= 59,84498705=> ce qui correspond à la valeur calculée par NRVQM (59,861) en tenant compte des arrondis.

Donc les valeurs de la métrique calculées par le plugin NRVQM correspondent bien aux valeurs prévues.

La figure 38 montre l'évolution de la valeur MOS avec une limitation de la bande passante à 600Kbit/s. On constate une période où les MOS calculés sont au maximum (95), puis on a une période où la limitation de la bande passante engendre des discontinuités à l'affichage qui changent la valeur du MOS calculé et puis on a une période où le MOS revient et reste à son maximum quand la limitation de la bande passante a été supprimée.
On peut remarquer que les premières discontinuités se présentent plusieurs secondes après qu'on a enclenché la limitation de la bande passante (à 10s). On peut également remarquer qu'après avoir supprimé la limitation de la bande passante (à 40s), plusieurs discontinuités se présentent encore. Ceci peut être expliqué par des buffers éventuels utilisés par des éléments GStreamer.

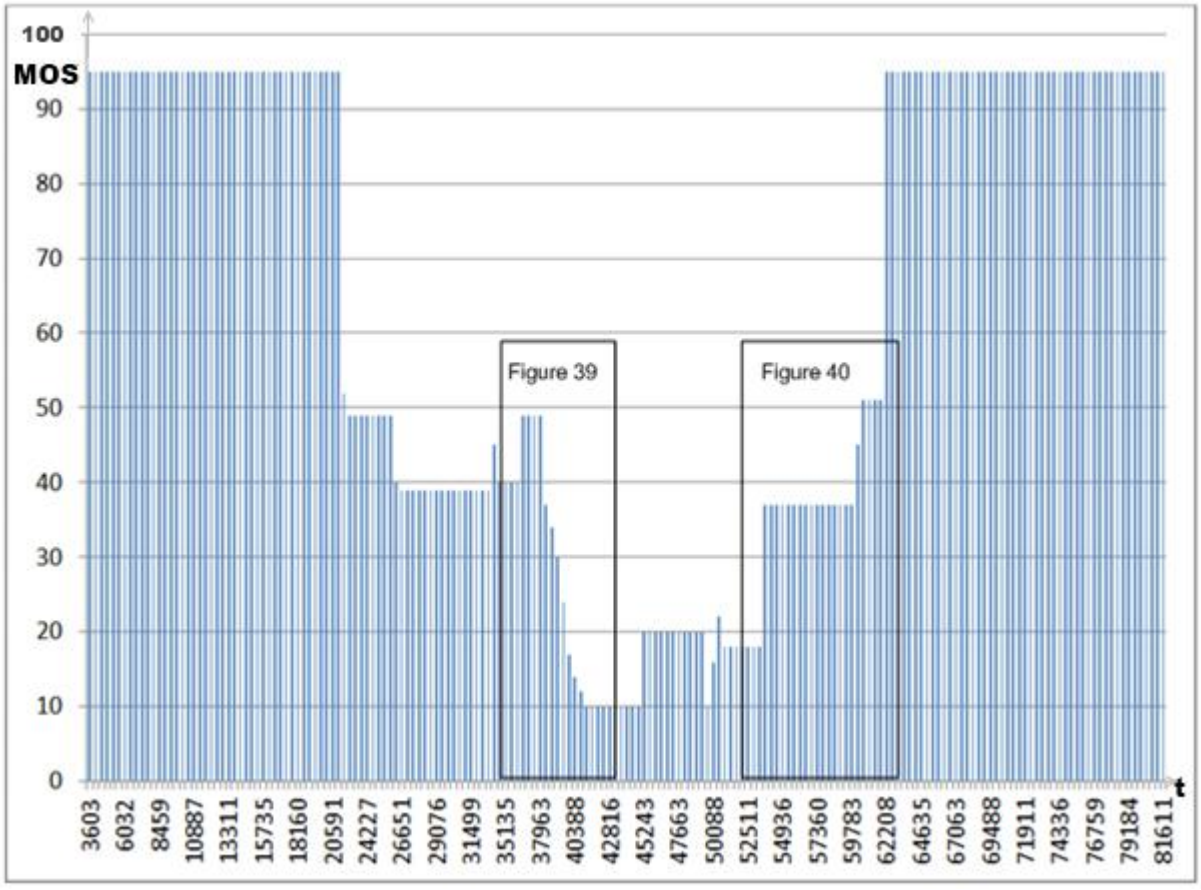


Figure 38 : Exemple d'évolution de la valeur MOS avec une limitation de la bande passante à 600Kbit/s.

Une deuxième constatation est que la valeur du MOS diminue si une discontinuité apparaît. La Figure 39 montre la variation du MOS calculé toutes les 400 ms pour 3 discontinuités (de 37872-39167, 39169-39529 et de 39970- 41537 ms) pour un extrait de la séquence. On constate bien que la valeur du MOS diminue à la fin des discontinuités (sauf si on est arrivé à la valeur minimale du MOS, 10, déterminé par [PASTRANA2005]).

Une autre constatation est que la valeur du MOS calculé diminue lorsqu'une discontinuité est en cours, mais pas encore terminée. Ceci correspond bien au fonctionnement spécifié.

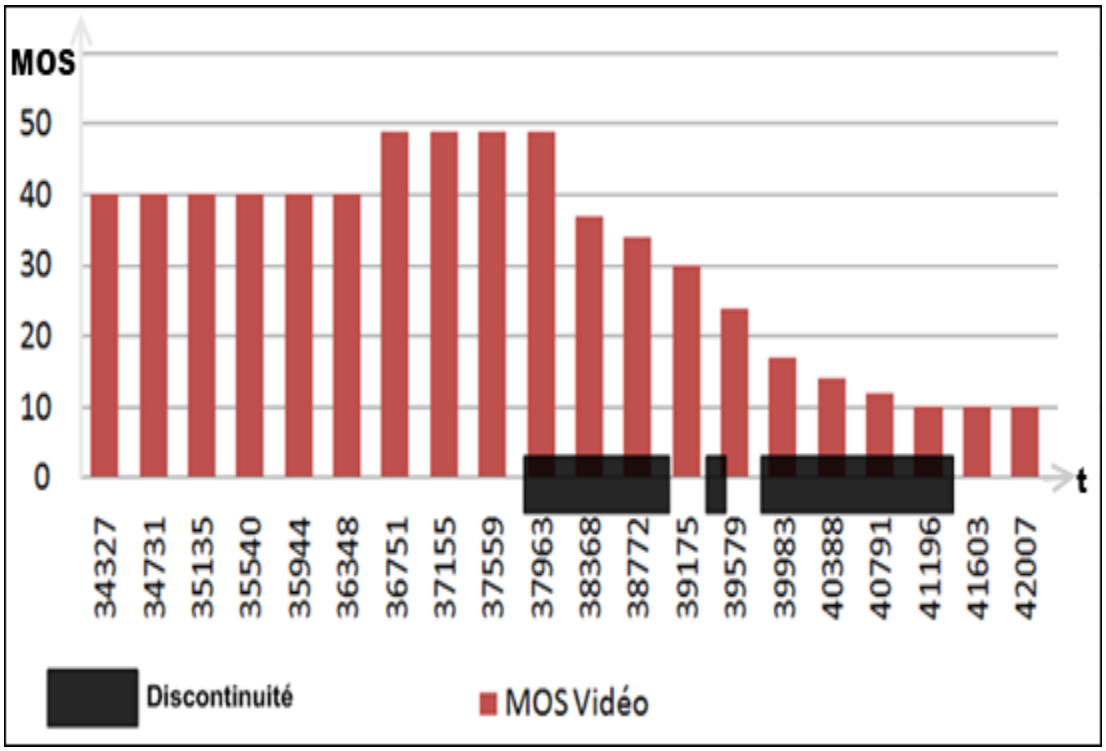


Figure 39: Exemple de variation du MOS calculé en fonction des discontinuités. Extrait d'une séquence.

La Figure 40 montre un exemple d'évolution du MOS après la dernière discontinuité qui s'est produite à 50919 ms et a duré 945ms, sur un extrait de la séquence vidéo. La fin de la discontinuité est donc 51864 ms. La fenêtre d'analyse étant de 10s, cette discontinuité sort de la fenêtre d'analyse à 61864. Le prochain calcul MOS se fait à 62208ms, où la valeur du MOS revient au maximum. Ceci correspond donc bien au fonctionnement voulu.

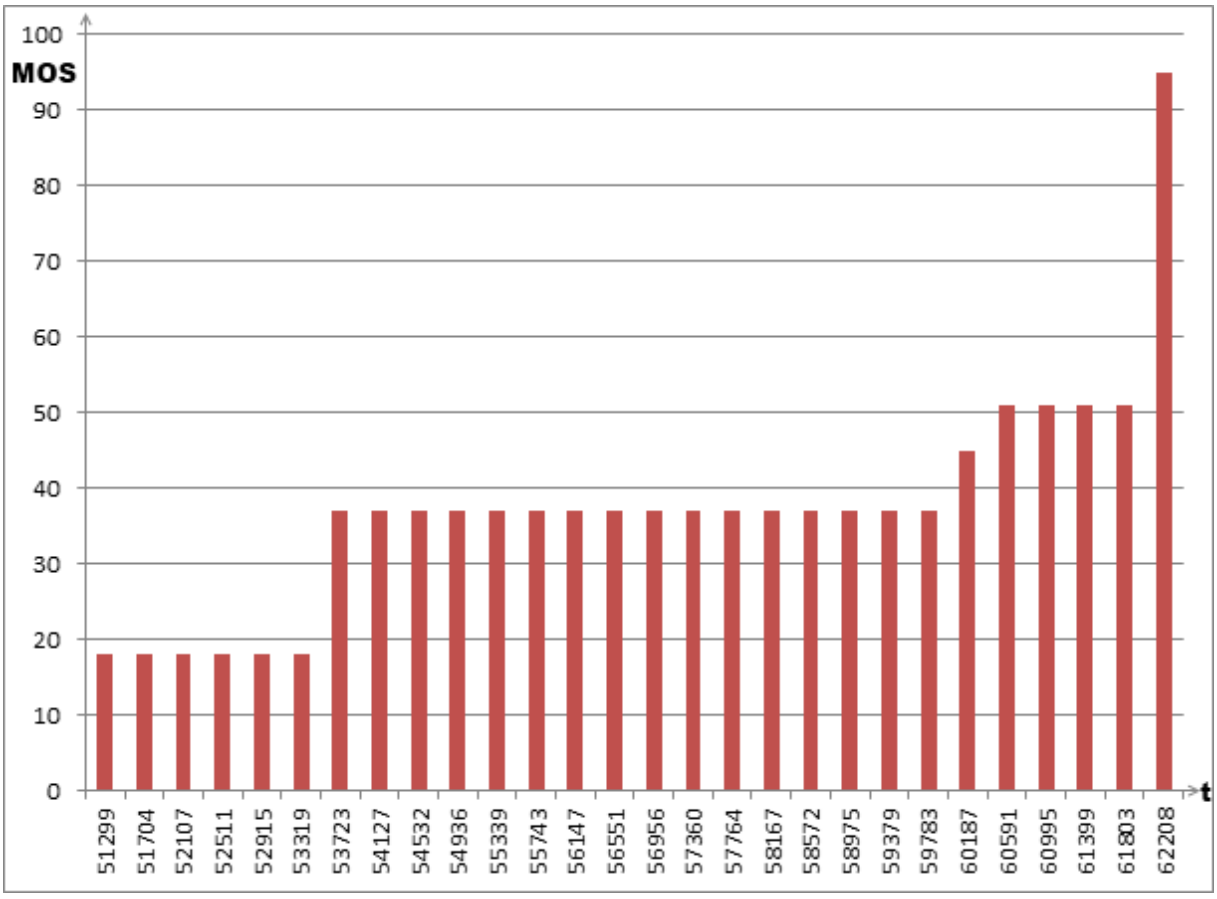


Figure 40: Exemple d'évolution du MOS calculé après la dernière discontinuité. Extrait d'une séquence.

Les Figures 41 et 42 permettent de comparer l'effet sur le MOS en considérant des discontinuités de durée différente. La Figure 41 montre un extrait d'une séquence vidéo avec 5 discontinuités de +/- 200ms de durée. On peut remarquer que la valeur MOS diminue jusque +/- 45. La Figure 42 montre un extrait d'une séquence vidéo avec 1 discontinuité de +/- 1050ms de durée. On peut remarquer que la valeur MOS diminue jusque +/- 50. Donc une longue discontinuité dégrade moins la qualité vidéo que plusieurs discontinuités de courte durée, ce qui correspond aux attentes de [PASTRANA2005].

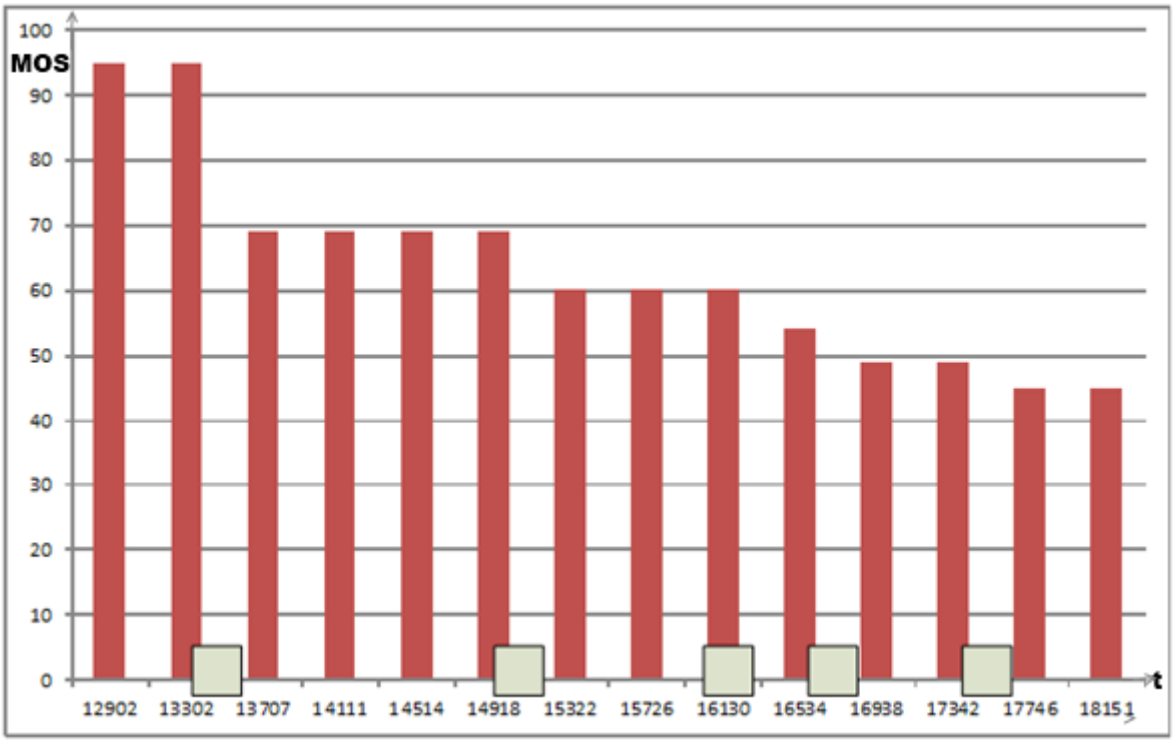


Figure 41 : Extrait d'une séquence d'évolution du MOS calculé pour 5 discontinuités de +/- 200 ms.

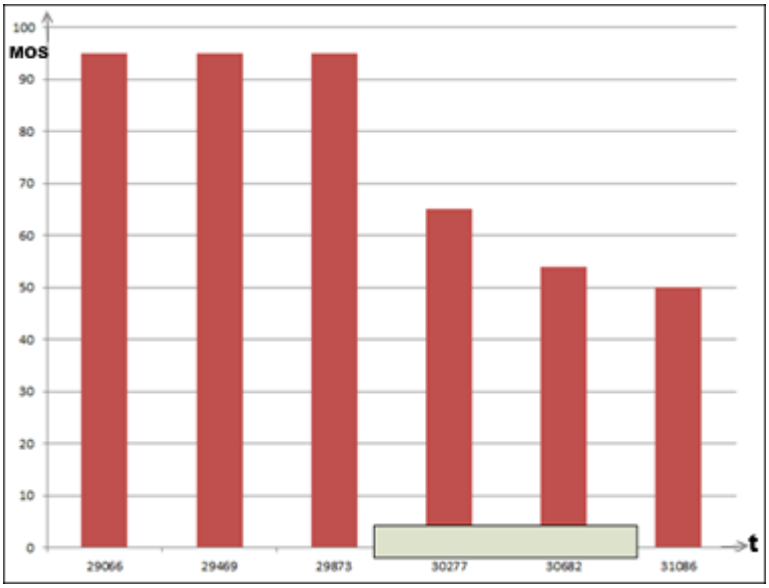
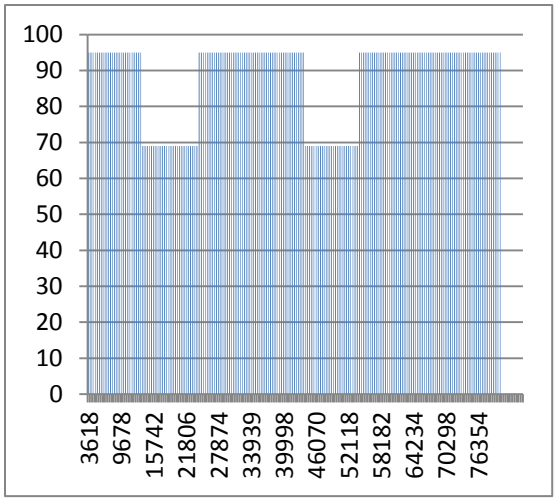
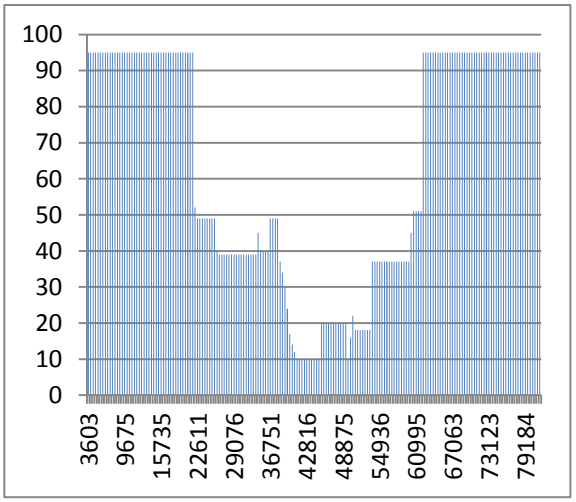


Figure 42 : Extrait d'une séquence d'évolution du MOS calculé pour 1 discontinuité de 1050 ms.

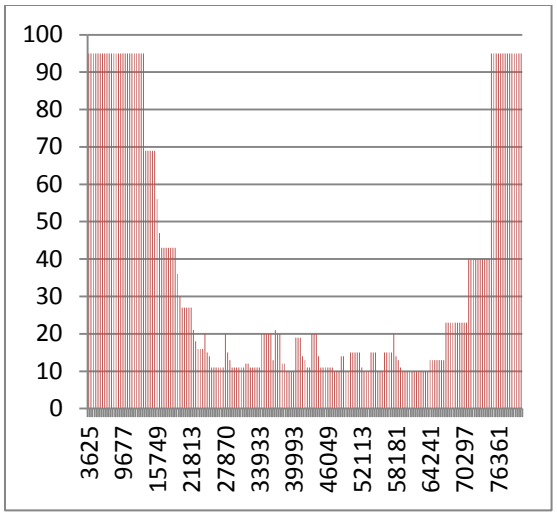
Pour les essais avec les autres bandes passantes, les résultats correspondent aux attentes : plus la bande passante est basse, plus les valeurs de la métrique MOS calculées sont basses. Voir les Figures 43 a-d.



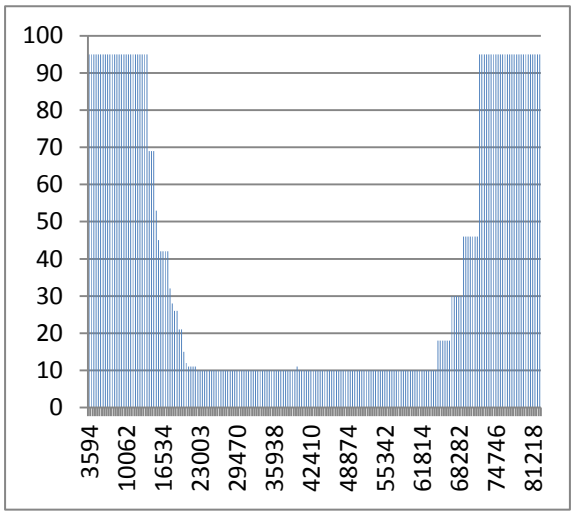
43.a: 800Kbit/s



43.b: 600Kbit/s



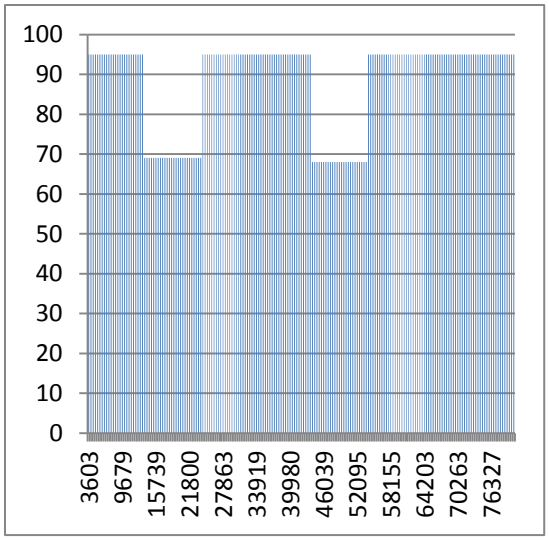
43.c: 500 Kbit/s



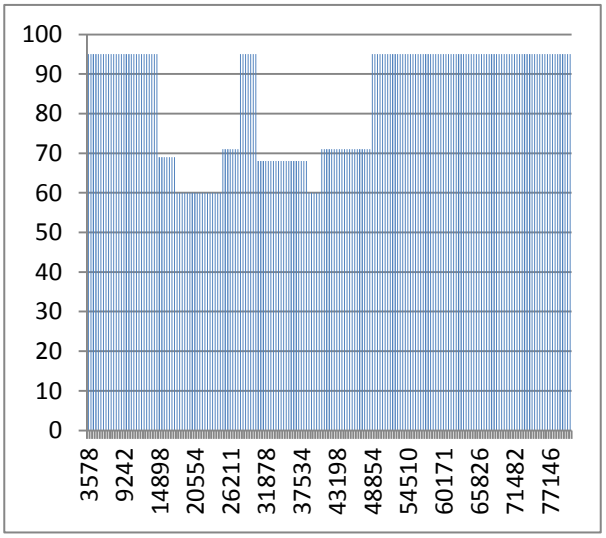
43.d: 400Kbit/s

Figure 43: Résultats de l'évolution de la valeur MOS calculée pour différentes bandes passantes.

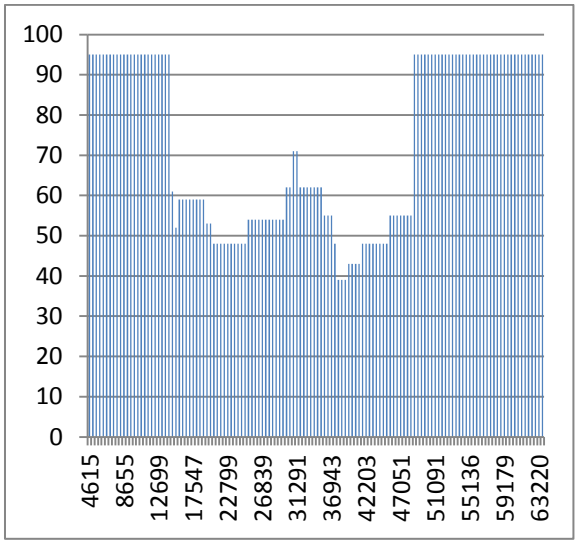
Les Figures 44 a-e montrent les résultats de la série de tests avec des taux de pertes de paquets différents. Les résultats sont comparables aux résultats des essais avec des bandes passantes limitées.



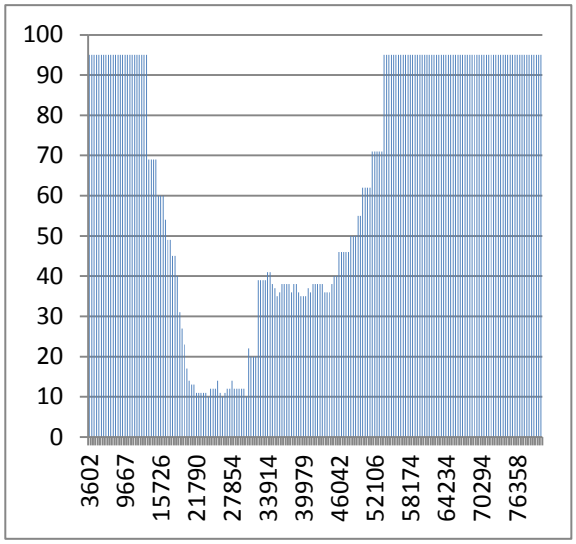
44.a: 0,1% de pertes



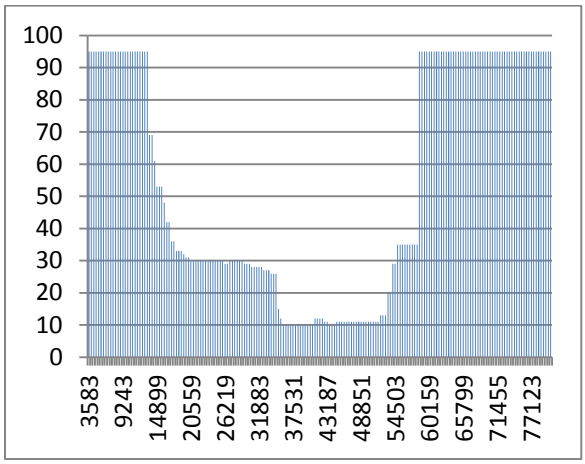
44.b: 0,5% de pertes



44.c: 1% de pertes



44.d: 2% de pertes



44.e: 5% de pertes

Figure 44 : Résultats de l'évolution de la valeur MOS calculée pour différents taux de pertes de paquets.

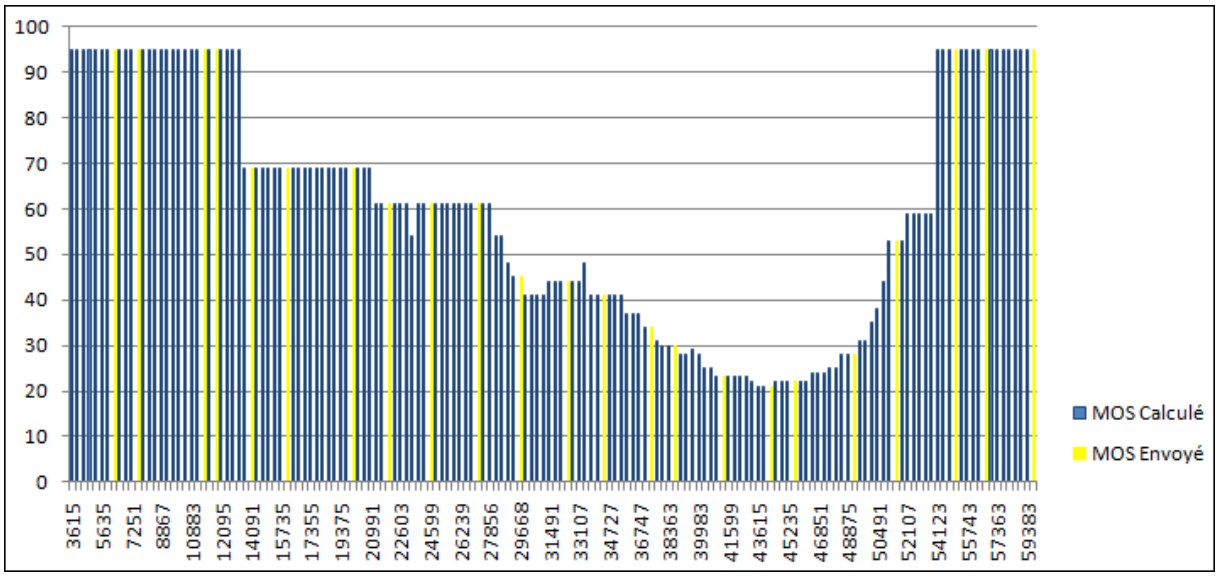
On peut ajouter que lors des essais effectués, d'autres types de dégradations se sont également manifestées. La dégradation de distorsion de bloc (Block distorsion) était très présente.

Résultats de l'essai sur le feed-back de la métrique MOS :

La Figure 45 montre l'évolution de la métrique MOS calculée et la valeur de la métrique MOS envoyée au serveur via les paquets RTCP XR QoE pour la séquence vidéo de test. On constate que le temps entre deux paquets RTCP peut varier, ce qui correspond bien au standard [RFC3550] (voir section 5.1.1). La valeur de la métrique MOS envoyée au serveur suit bien la valeur calculée. On constate également que la valeur du MOS calculée peut varier dans la période entre deux paquets RTCP.

On remarque de nouveau que la valeur MOS calculée change seulement un certain temps après le début de la perturbation (perte de paquets de 1%). La perturbation a démarrée à 10s, la première discontinuité se présente vers 13515ms et dure 235 ms. La valeur MOS calculée vers 13712ms est alors de 69. Donc on a un retard de 3712ms entre le début de la perturbation, et son implication sur la valeur MOS calculée. Ce retard est probablement dû à l'utilisation des buffers. L'implication de la perturbation sur la valeur MOS envoyé au serveur via le paquet RTCP est constatée au prochain paquet RTCP envoyé, c'est-à-dire vers 14091ms.

On peut en conclure que les conséquences dues à une perturbation ne seront connues du serveur de streaming que plusieurs secondes après son apparition. Ceci est dû à la fréquence d'envoi des paquets RTCP suivant le standard [RF33550] et au retard créé par l'utilisation de buffers par des éléments GStreamer.



Figire 45 : Evolution de la métrique MOS calculée/envoyée.

La Figure 46 montre le contenu d'un paquet RTCP XR QoE renvoyé au serveur intercepté par le 'sniffer' Wireshark. La structure du paquet correspond à celui de la Section 5.1.4. La valeur Hex '2E3E' tout à la fin correspond à la valeur de la métrique MOS renvoyée à ce moment, cette valeur correspond à la valeur MOS calculée (qui était de 46,2444) en représentation integer scaled 8:8. Le '2E' en Hex correspond au 46 et le '3E' correspond à la représentation de 0,2444. En effet 0,2444 X 256 = 62,5664, la partie entière étant de 62 ou 3E Hex. Les autres valeurs des champs sont pour le BT on a Hex '64' = 100 decimal, longueur = 1, Type = '0100' en binaire, Calc alg = 4. Dir, Chan, I sont 0. Ces valeurs correspondent bien aux valeurs spécifiées.
La Figure 46 montre également la structure du paquet RTCP composé. Ici le paquet composé contient un paquet RTCP RR, un paquet RTCP SDES et le paquet RTCP XR QoE.

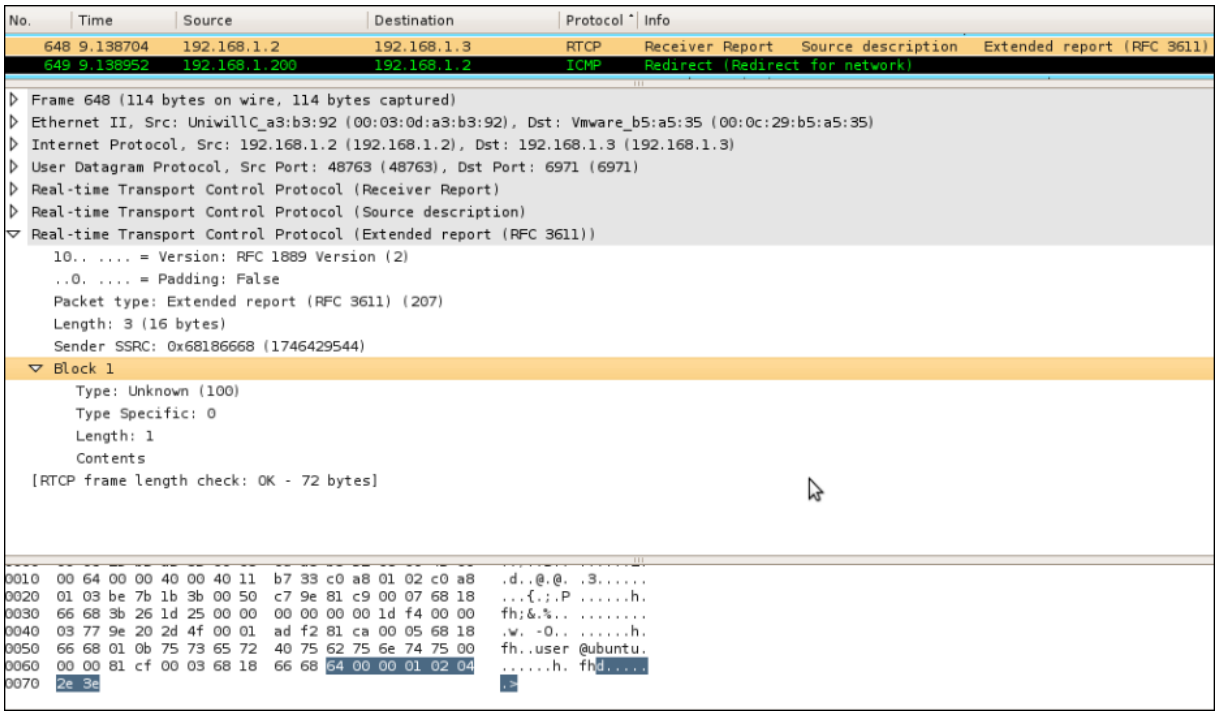


Figure 46: Contenu du paquet RTCP XR QoE renvoyé au serveur.

9.1.4 Conclusion

Les essais ont montré que la solution proposée détecte les discontinuités, calcule la métrique MOS et renvoie un paquet RTCP XR QoE avec la valeur de la métrique calculée comme on l'avait spécifié.
Comme prévu, les discontinuités qui ne sont pas encore terminées sont prises en considération pour le calcul du MOS, mais ceux dont la fin est en dehors de la fenêtre d'analyse ne le sont pas.
On a constaté à plusieurs reprises que des discontinuités apparaissent plusieurs secondes après une perturbation (perte de paquets, erreur de paquets, bande passante limitée), dues probablement à l'utilisation de buffers. Ceci retarde le renvoi de la métrique MOS vers le serveur et restreint l'utilisation de la solution proposée dans une optique d'adaptation de la

qualité vidéo en ‘temps réel’, même si la fréquence d’envoi des paquets RTCP sera augmentée.

On peut également noter que le framework GStreamer, ou quelques'un de ses éléments, n'arrivent pas à rétablir leur fonctionnement normal après des perturbations de communication lorsqu'on utilise des vidéos encodées avec des débits binaires faibles.

10 Conclusion

Pour l'étude des possibilités d'adaptation de la qualité vidéo, on a d'abord déterminé les spécificités du streaming de la vidéo mobile transmise par les protocoles RTP/RTCP/RTSP:

-bande passante variant de 180kbps à 7,6 Mbps
-perte de paquets et erreurs de bit possibles
-résolution vidéo QCIF/CIF/QVGA
-frame rate de 5-15 images par seconde
-codec de la famille MPEG (surtout H.263 et H.264)
-performance des terminaux mobiles souvent limitée

Lors de l'étude de l'évaluation de la qualité vidéo, on a déterminé que la métrique calculée en utilisant un modèle mathématique (modelé VQM) doit avoir une bonne corrélation avec le MOS (*Mean Opinion Score*). Le MOS provient de l'évaluation de la qualité subjective de la vidéo. Une possibilité de réaliser cela est d'utiliser un modèle qui prend en compte la perception humaine de la vidéo décodée (Métrique de l'image).
Dans le contexte du streaming vidéo vers un terminal mobile, l'utilisation d'une métrique NR (non référencée) s'avère la plus avantageuse, puisqu'elle utilise moins de bande passante.
La dégradation image "jitter/jerkiness", qui correspond à un mouvement saccagé de la vidéo, peut avoir une grande influence sur la valeur MOS lorsqu'il y a une perte de paquets.

Ceci nous amène aux critères de sélection pour les modèles VQM pour notre cas d'étude:
-modèle disposant d'une bonne corrélation par rapport au MOS.
-modèle tenant compte d'une bande passante limitée
-modèle adapté aux résolutions vidéo mobiles
-modèle tenant compte du « *jerkiness/jitter* »
-modèle nécessitant une puissance de calcul limitée

On a présenté plusieurs modèles VQM qui utilisent des concepts différents. Les critères déterminés précédemment nous ont permis de sélectionner le modèle basé sur la rupture de la fluidité vidéo/netteté [PASTRANA2005], [PASTRANA2006], [PASTRANA2007]. Il s'agit d'un modèle VQM NR, de type métrique de l'image. Il prend en compte la dégradation "*jitter/jerkiness*" et a été testé dans des scénarios typiques pour le streaming vidéo mobile en considérant les résolutions, bandes passantes typiques. Pour améliorer davantage la correspondance au critère de la puissance de calcul limitée, on choisit de dissocier le VQM fluidité/netteté et de n'utiliser que la métrique fluidité du VQM fluidité/netteté.

L'analyse du feed-back MOS via les protocoles RTCP/RTSP a montré qu'il n'existe pas encore de méthode standardisée pour renvoyer la métrique MOS du client vers le serveur de streaming. On a présenté quatre possibilités qui étendent/adaptent les protocoles RTCP/RTSP existants pour réaliser le feed-back. On a sélectionné la méthode proposée par

[CLARK et al, 2008], qui se base sur un nouveau bloc de rapport RTCP XR, le bloc QoE (*Quality of Experience*). Cette méthode permet de renvoyer la métrique MOS tout en se basant sur la technique RTCP standard. Ceci permet de limiter le travail d'implémentation au niveau du client (et par après au niveau du serveur). On a également vu que le protocole RTCP n'est pas approprié aux cas d'utilisation nécessitant un feed-back immédiat et fiable, comme l'adaptation de la qualité vidéo lors des variations brusques de la bande passante. Pour une adaptation de la bande passante sur une échelle de temps de plusieurs secondes, le RTCP peut être utilisé.

Le modèle VQM basé sur la rupture de la fluidité étant initialement utilisé sur des séquences de vidéo de 10s de durée, on a adapté le modèle pour tenir compte de contenus vidéo continus de durée variable. L'idée est de pouvoir calculer le MOS à n'importe quel moment en utilisant la fenêtre d'analyse qui se déplace avec le temps.

On a alors intégré les différents composants sélectionnés et on a proposé l'architecture logique de la solution générale pour la partie client d'un système d'adaptation automatique de la qualité d'un service de streaming vidéo sur base du client média GStreamer.
Deux fonctionnalités sont ajoutées au client média :

-le calcul de la métrique MOS, qui sera réalisé par un nouveau plugin GStreamer

-le feed-back du MOS via le protocole RTCP XR QoE, qui sera réalisé par une adaptation des plugins GStreamer RTP/RTSP existants.

L'implémentation du modèle VQM, pour le calcul de la métrique MOS, dans le nouveau plugin GStreamer NRVQM, a nécessité plusieurs décisions :
-la technique de détection d'une discontinuité proposée par [PASTRANA2005] n'étant pas possible dans le cas du framework GStreamer, on a décidé d'implémenter la détection d'une discontinuité en utilisant la différence de temps entre deux images. Ceci nous a amené à augmenter le seuil perceptuel de détection des discontinuités à 200ms, pour prendre en compte les vidéos avec une fréquence d'affichage de 5-15 FPS. Ceci diminue la précision de la valeur MOS calculée, qui ne prend plus en compte les discontinuités de moins de 200ms.
- La signalisation de la métrique MOS entre les éléments GStreamer NRVQM et GstRTSPSrc est implémentée par un envoi périodique (400ms) d'un évènement, contenant la métrique MOS calculée, sur le bus GStreamer.

La technique de détection de discontinuités implémentée dans le Plugin NRVQM ne nécessite qu'une puissance de calcul faible. Le calcul de la métrique n'a besoin que de quelques opérations et donc nécessite seulement une puissance de calcul faible. L'implémentation de la solution proposée sur un client terminal mobile ne va pas fortement augmenter la puissance de calcul nécessaire. Si le terminal mobile dispose d'une puissance de calcul suffisante pour décoder la vidéo H.264, il devrait être capable de faire tourner la

solution proposée. (En faisant l'hypothèse que le framework GStreamer et tous les plugins GStreamer nécessaires peuvent être installés sur le terminal mobile).

On a adapté les éléments GStreamer GstRTSPSrc, GstRtpBin, GstRtpSession et le module RtpSession des plugins GStreamer qui s'occupent de la communication RTP/RTSP pour réaliser le feed-back de la métrique MOS vers le serveur. Pour l'implémentation du bloc de rapport RTCP XR QoE, on a choisi de ne pas utiliser le champ "*Tag*" qui identifie un bloc d'identification de mesure qui n'est pas absolument nécessaire dans notre contexte.

On a proposé un environnement de test pour réaliser des essais de fonctionnement sur la solution proposée. L'analyse des résultats a montré que la valeur calculée de la métrique MOS correspond bien aux prévisions. Les discontinuités en cours et terminées sont correctement prises en considération. Le feed-back de la métrique MOS vers le serveur via le paquet RTCP est correctement effectué.
Donc la solution proposée correspond bien aux fonctionnalités spécifiées.
On a constaté un retard de plusieurs secondes entre le début d'une perturbation (bande passante limitée) et l'apparition de la dégradation qui en résulte. Ce retard est probablement dû aux buffers utilisés par les éléments GStreamer.

Dans ce mémoire, on s'est limité à un cas de streaming unicast. Dans cette optique on a simplifié la méthode proposée par [CLARK et al, 2008] en supprimant le bloc d'identification de mesure. Cela ne correspond pas aux lignes directrices de [OTT et al, 2010].
On devra tenir compte de ces points lors d'une intégration éventuelle de la solution client proposée avec la partie serveur.

Dans le cadre de ce mémoire, l'étude était orientée vers la partie client de l'adaptation de la qualité du streaming vidéo. La partie serveur n'étant pas encore réalisée, il n'était pas possible de tester le fonctionnement complet de l'adaptation de la qualité vidéo et de vérifier l'amélioration de l'expérience de l'utilisateur.
On peut quand même en tirer plusieurs conclusions :

-Le calcul de la métrique MOS et son feedback vers le serveur fonctionne comme spécifié. On a quand même dû faire un compromis sur le choix du seuil de perception d'une discontinuité, qui a une influence sur la précision de la valeur MOS calculée et de la fréquence d'affichage nécessaire à une vidéo.

-D'autres types de dégradations ont pu être observés lors des essais, entre autres la distorsion en bloc. Afin d'améliorer la métrique calculée MOS, il peut être intéressant d'étendre le plugin NRVQM avec d'autres métriques VQM qui prennent compte ces dégradations, dans l'avenir.

-Suite aux retards, principalement engendrés par les buffers Gstreamer et la fréquence de renvoi des paquets RTCP, la solution n'est pas adaptée aux cas d'utilisation

nécessitant un feed-back immédiat. Elle peut être utilisée pour réaliser une adaptation de la bande passante sur une échelle de temps de plusieurs secondes.

Une piste à suivre est d'étudier la partie serveur de ce système, d'intégrer la présente étude et proposer une solution globale pour le serveur et le client. Ceci permettra alors de vérifier le fonctionnement du système complet, et partant, l'amélioration de l'expérience de l'utilisateur.

11 Bibliographie

[3GPP2006] 3rd Generation Partnership Project, “Technical Specification Group Serices System Aspects, Transparent end-to-end Packet-switched Streaming Services (PSS), Protocols and codecs (Release 6)”, 3GPP TS 26.234 V6.9.0 (2006-09). http://www.3gpp.org.

[ARM2009] ARM Architecture. http://en.wikipedia.org/wiki/ARM_architecture, dernière viste le28/0/2009.

[APPLE2010]Http Live Streaming, Networking&Internet. http://developer.apple.com/iphone/library/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/StreamingMediaGuide.pdf

[ATIS2006] QoSM Task Force. ATIS-0800004, “A Framework for QoS Metrics and Measurements Supporting IPTV Services”, Alliance for Telecommunications Industry Solutions. Washington, 2006.

[BABU2004]Babu R. V., Bopardikar A. S., Perkis A., Hillestad O. I. ,” NO-REFERENCE METRICS FOR VIDEO STREAMING APPLICATIONS”, http://www.q2s.ntnu.no/~hillesta/papers/Babu_PacketVideo_2004.pdf. Dernière visite le 18/08/2009.

[BELGACOM 2009] Aperçu des réseaux data mobiles. http://customer.proximus.be/FAQ/topic.jsp?language=fr&contentpath=00df7d283fe7496000000115c018b9c6#HSDPA_POS_02. Dernière visite le 18/08/2009.

[BIRNEY2003] Birney Bill, “Intelligent Streaming. Microsoft Corporation”. http://www.microsoft.com/windows/windowsmedia/howto/articles/intstreaming.aspx.(Dernière visite 17/04/2010).

[BLACK1996] M. J. Black and P. Anandan, “The Robust Estimation of Multiple Motions: Parametric and Piecewise-Smooth Flow Fields”, Computer Vision and Image Understanding 63, pp. 75–104, Jan. 1996. Dans [LU2007]

[BOOK2000] Book M., “Resource Reservation Protocol Realtime Transport Protocol”, Lehrstuhl für Elektronische Systeme und Vermittlungstechnik, Fakultät für Elektrotechnik, Universität Dortmund. http://www.matthiasbook.de/papers/rsvp-rtp/files/rsvp-rtp-paper.pdf, Dernière visite 27/04/2010.

[BOULTON et al, 2010] Boulton R.J., Walthinsen E., Baker S., Johnson L., Bultje R.,Kost S., Maeller T., GStrealer Plugin Writer's Guide", http://gstreamer.freedesktop.org/data/doc/gstreamer/head/pwg/pwg.pdf. Dernière visite, le 28/04/2010.

[BOYCE1998] J. Boyce and R. Galianello, "Packet loss effects on MPEG video sent over public internet", in ACM International Multimedia Conference, 1998, pp. 181–190.in [BABU2004]

[BT2003] "H.264 – the new video codec. BT Exact's leading-edge implementation", http://www.ipvalue.com/technology/docs/BT_H264.pdf. Dernière visite le 27/05/2010.

[CERMAK2005] G. W. Cermak, "Packet loss, Bandwith, and latency affect judged quality of videoconferencing", http://enpub.fulton.asu.edu/resp/vpqm2005/papers/221.pdf. Dernière visite le 25/05/2010.

[CLARK et al,2008] Clark Alan, Hunt Geoff, "RTCP XR Report Block for QoE Metrics Reporting", AT Working Group Internet Draft, draft-ietf-avt-rtcp-xr-qoe-00. http://tools.ietf.org/id/draft-ietf-avt-rtcp-xr-qoe-00.txt.

[DIMOU2005] A. Dimou, O. Nemethova, M. Rupp, "Scene Change Detection for H.264 Using Dynamic Threshold Techniques," in Proc. of the 5th EURASIP Conference on Speech and Image Processing, Multimedia Communications and Service, Smolenice, Slovac Republic. Jul. 2005. Dans [RIES2008]

[Feamster2002] N. Feamster and H. Balakrishnan, "Packet loss recovery for streaming video.," in International Packet Video Workshop, April 2002. In [BABU2004]

[FGIPTV2007]ITU, "Focus Group On IPTV, FG IPTV-C-0769", http://www.itu.int/md/T05-FG.IPTV-C-0769/fr. Dernière visite, le 30/08/2009.

[HEYNA2003] Heyna A, Briede M, Schmidt U , „Datenformate im Medienbereich", Fachbuchverlag Leipzig. Carl Hanser Verlag München Wien, 2003.

[HOROWITZ et al, 2003] M. Horowitz, A. Joch, F. Kossentini, A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis", dans IEE Transactions on circuits ans systems for video technology, Vol. 13, No.7, July 2003. http://lts4www.epfl.ch/teaching/ic/repository/01218201.pdf Dernière visite le 27/05/2010.

[HUNT et al, 2009] Hunet G., Clark A., « RTCP XR Report Block for Measurement Identity", Internet-Draft, http://tools.ietf.org/html/draft-ietf-avt-rtcp-xr-meas-identity-02. Dernière visite le 13/05/2010.

[Huynh-Thu2006] Huynh-Thu Q., Ghanbari M., „IMPACT OF JITTER AND JERKINESS ON PERCEIVED VIDEO QUALITY", http://*enpub.fulton.asu.edu/resp/vpqm2006/papers06/308.pdf.* Dernière visite le 29/08/2009.

[Huynh-Thu2007] Huynh-Thu Q., Brotherton M., Hands D., Brunnström K., Ghanbari M., "EXAMINATION OF THE SAMVIQ METHODOLOGY FOR THE SUBJECTIVE ASSESSMENT OF MULTIMEDIA QUALITY", http://enpub.fulton.asu.edu/resp/vpqm2007/papers/400.pdf. Dernière visite le 29/08/2009**.**

[Jammeh et al, 2002] Jameh E., Paredes-Farrera M., Ghanbari M.,"Transporting real time transcoded Video Over the Internet using end-to-end control", http://research.microsoft.com/en-us/um/redmond/events/pv2002/papers/38-hasexghimx.pdf

[LI2007] Li Jianjun,"Complexity reduction for HW implementation of H.264/AVC", Research Centre for Integrated Microsystems, Electrical and Computer Engineering, Universuty of Windwor. http://www.vlsi.uwindsor.ca/presentations/2007/3-Complexity%20Rdeuction.pdf. Dernière visite le 27/05/2010.

[LIU2008] Xingang Liu, Yeol-Kook Yoo, "Real-Time Reference-Free Video Quality easurement for Multimedia Communication," icess, pp.69-76, 2008 International Conference on Embedded Software and Systems, 2008.

[LU2007] LU Z., Lin W, Seng B. C. , Kato S., Ong E., Yao S. , "PERCEPTUAL QUALITY EVALUATION ON PERIODIC FRAME-DROPPING VIDEO", Image Processing, 2007.ICIP 2007.IEEE International Conference on.Volume3.

[MARZILANO2002] Pina Marzilano etc, "A no-reference perceptual blur metric", *Proceedings of the International Conference on Image Processing*, vol. 3 (2002), p. 57-60 2002. Dans [Liu2008]

[Microsoft2009] Windows Media Video 9 Series Codecs, http://www.microsoft.com/windows/windowsmedia/forpros/codecs/video.aspx#WindowsMediaVideo9AdvancedProfile

[MULROY et al, 2009] Mulroy P., Appleby S., Nilsson M., Crabtree B.," THE USE OF MULTCP FOR THE DELIVERY OF EQUITABLE QUALITY VIDEO", BT Innovate, Broadband

Applications Research Centre. http://research.microsoft.com/en-s/um/redmond/events/pv2009/ papers/session_congestion_control/paper%2018.pdf

[OTT et al, 2010] Ott J., Perkins C., "Guidelines for Extending the RTP Control Protocol (RTCP)", AVT Working Group, Internet-Draft, http://tools.ietf.org/html/draft-ietf-avt-rtcp-guidelines-04. Dernière visite le 19/05/2010.

[PASTRANA2004] R.R. Pastrana-Vidal, C. Colomes, J.C. Gicquel, H. Cherifi, » Métrique perceptuelle de rupture de fluidité vidéo sans référence »,France Télécom R&D, DIH/EQS/M@I.http://www-rech.telecom-lille1.eu/coresa2004/articles/p204-pastrana.pdf. Dernière visite le 09/05/2010

[PASTRANA2005] Ricardo Rafael Pastrana-Vidal, « Vers uneMétrique Perceptuelle de Qualité Audiovisuelle dans un Contexte à Service Non Garanti », Ph.D. thesis, Université de Bourgogne, 2005.

[PASTRANA2006]Pastrana-Vidal R.,Gicquel J.," AUTOMATIC QUALITY ASSESSMENT OF VIDEO FLUIDITY IMPAIRMENTS USING A NO-REFERENCE METRIC", http:// enpub.fulton.asu.edu/resp/vpqm2006/papers06/311.pdf. Dernière visite le 29/08/2009.

[PASTRANA2007] Pastrana-Vidal R.,Gicquel J., "A NO-REFERENCE VIDEO QUALITY METRIC BASED ON A HUMAN ASSESSMENT MODEL", *http://enpub.fulton.asu.edu/resp/vpqm2007/papers/403.pdf.* Dernière visite le 29/08/2009.

[RAMIN2008] Ramin N., Pastrana-Vidal R., Saadane H., "No reference video quality assessement metrics for multimedia : state of the art of signal-based approaches", http://portal.etsi.org/docbox/Workshop/2008/2008_06_STQWORKSHOP/ORANGE_NicolasRamin.pdf. Dernière visite le 29/08/2009.

[RFC2326] Schulzrinne H., Rao A., Lanphier R., "Real Time Streaming Protocol (RTSP)", The internet society. RFC 2326. http://www.ietf.org/rfc/rfc2326.txt. Dernière visite le 25/04/2010**.**

[RFC3550] Schulzrinne H., Casner S., Frederick R., Jacobson V., RTP ," A Transport Protocol for Real-Time Applications", The internet Society. RFC 3550. http://www.ietf.org/rfc/rfc3550.txt. denière visite le 25/04/2010.

[RFC3611] Friedman T., Caceres R., Clark A., “RTP Control Protocol Extended Reports (RTCP XR)”, RFC 3611. http://www.rfc-editor.org/rfc/rfc3611.txt. Dernière visite le 25/04/2010.

[RIES2008] Ries M., Nemethova O., Rupp M.,”Video Quality Estimation for Mobile H.264/AVCVideo Streaming”, Journal of communications, VOL3.NO.1, January 2008.

[RIES2007] Ries M., Nemethova O., Rupp M.,” Motion Based Refernce-Free Quality Estimation for H.264/AVC Video Streaming”, http://publik.tuwien.ac.at/files/pub-et_12118.pdf.

[ROSSHOLM2008] Andreas Rossholm, Benny Lövström,” A New Video Quality Predictor Based on Decoder Parameter Extraction”, Inst Syst & Technologies Informat, Control & Commun. Oporto 2008.

[SCHIERL et al, 2004] Schierl Thomas, Wiegand Thomas,” H.264/AVC rate adaptation for internet streaming”, Frauenhofer Institute for Telecommunications – Heinrich Hertz Institute (HHI). http://ip.hhi.de/imagecom_G1/assets/pdfs/pv04_rate_adaptation.pdf.

[SNELL2009] MPEG Encoding Basics, http://www.media-matters.net/docs/resources/ Digital%20Files/MPEG/ MPEG%20Encoding%20Basics.pdf

[TAYMANS et al, 2010] Tayman W., Baker S., Wingo A., Bultje R., Kost S., “GStreamer Application Development Manual (0.10.29)”, http://gstreamer.freedesktop.org/data/doc/ gstreamer /head/manual/manual.pdf. Dernière visite 28/04/2010.

[VERSCHEURE1999] O. Verscheure, P. Frossard, and M. Hamdi, “User-oriented QoS analysis in MPEG-2 delivery,” *Real-Time Imaging*, vol. 5, no. 5, pp. 305–314,1999. Dans [WINKLER2008]

[VIDEOLAN] “VLC media player”, http://www.videolan.org/. Denière visite , le 2/05/2010.

[WALCOMO] Worthy visuAL COntent on Mobile, http://recherche-technologie.wallonie.be/projets/index.html?IDD=8442

[WANG2002] Wang Z., Sheikh H.R., Bovik A.C.,” No-Reference Perceptual Quality Assessment of JPEG Compressed Images”, http://www.cns.nyu.edu/~zwang/files/papers/icip02.html. Dernière visite le 29/08/2009.

[WEN et al, 2006] Wen Sun, Yan Lu, Feng Wu," BIT-STREAM SWITCHING IN MULTIPLE BIT-RATE VIDEO STREAMING USING WYNER-ZIV CODING", http://research.microsoft.com/en-us/people/fengwu/switch_icme_06.pdf

[WATTMORGAN1983] R. J. Watt and M. J. Morgan, "The recognition and representation of edge blur: Evidence for spatial primitives in human vision," *Vision Res.*, vol. 23, no. 12, pp. 1465–1477, 1983. Dans [PASTRANA2007].

[Winkler2008] Winkler S., Mohandas P.," The Evolution of Video Quality Measurement: From PSNR to Hybrid Metrics", IEEE Trans. Broadcasting VOL. 54, NO. 3, Septmeber2008.

12 Annexes

Annexe A

Voici le déroulement d'un exemple de session RTSP complète [RFC2326].

Client C demande un média des serveurs de média A (audio.example.com) et V (video.example.com). La description de média se trouve sur le serveur W.

```
C->W: GET /twister.sdp HTTP/1.1
        Host: www.example.com
        Accept: application/sdp

    W->C: HTTP/1.0 200 OK
        Content-Type: application/sdp
        v=0
        o=- 2890844526 2890842807 IN IP4 192.16.24.202
        s=RTSP Session
        m=audio 0 RTP/AVP 0
        a=control:rtsp://audio.example.com/twister/audio.en
        m=video 0 RTP/AVP 31
        a=control:rtsp://video.example.com/twister/video

    C->A: SETUP rtsp://audio.example.com/twister/audio.en RTSP/1.0
        CSeq: 1
        Transport: RTP/AVP/UDP;unicast;client_port=3056-3057

    A->C: RTSP/1.0 200 OK
        CSeq: 1
        Session: 12345678
        Transport: RTP/AVP/UDP;unicast;client_port=3056-3057;
                server_port=5000-5001

    C->V: SETUP rtsp://video.example.com/twister/video RTSP/1.0
        CSeq: 1
        Transport: RTP/AVP/UDP;unicast;client_port=3058-3059

    V->C: RTSP/1.0 200 OK
        CSeq: 1
        Session: 23456789
        Transport: RTP/AVP/UDP;unicast;client_port=3058-3059;
                server_port=5002-5003

    C->V: PLAY rtsp://video.example.com/twister/video RTSP/1.0
        CSeq: 2
        Session: 23456789
        Range: smpte=0:10:00-

    V->C: RTSP/1.0 200 OK
```

CSeq: 2
Session: 23456789
Range: smpte=0:10:00-0:20:00
RTP-Info: url=rtsp://video.example.com/twister/video;
seq=12312232;rtptime=78712811

C->A: PLAY rtsp://audio.example.com/twister/audio.en RTSP/1.0
CSeq: 2
Session: 12345678
Range: smpte=0:10:00-

A->C: RTSP/1.0 200 OK
CSeq: 2
Session: 12345678
Range: smpte=0:10:00-0:20:00
RTP-Info: url=rtsp://audio.example.com/twister/audio.en;
seq=876655;rtptime=1032181

C->A: TEARDOWN rtsp://audio.example.com/twister/audio.en RTSP/1.0
CSeq: 3
Session: 12345678

A->C: RTSP/1.0 200 OK
CSeq: 3

C->V: TEARDOWN rtsp://video.example.com/twister/video RTSP/1.0
CSeq: 3
Session: 23456789

V->C: RTSP/1.0 200 OK
CSeq: 3

Annexe B

L'annexe B reprend le code source du plugin NRVQM. Les fichiers utilisés sont gstnrvqm.c et gstnrvqm.h.

B.1 Le fichier source gstnrvqm.c

```
/* GStreamer
* Copyright (C) <1999> Erik Walthinsen <omega@cse.ogi.edu>
* Copyright (C) <2003> David Schleef <ds@schleef.org>
* Copyright (C) 2003 Arwed v. Merkatz <v.merkatz@gmx.net>
* Copyright (C) 2006 Mark Nauwelaerts <manauw@skynet.be>
*
* This library is free software; you can redistribute it and/or
* modify it under the terms of the GNU Library General Public
* License as published by the Free Software Foundation; either
* version 2 of the License, or (at your option) any later version.
```

```
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the
 * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */

/*
 * This file was (probably) generated from
 * gstvideotemplate.c,v 1.12 2004/01/07 21:07:12 ds Exp
 * and
 * make_filter,v 1.6 2004/01/07 21:33:01 ds Exp
 */

/**
 * SECTION:element-nrvqm
 *
 * Calcule le MOS Vidéo suivant un algorithme basé sur
 * l'algorithme de la métrique de fluidité de Pastrana/Vidal
 * et envoie toutes les 500 ms un évenement "video/vqm"
 * sur les bus gstreamer avec la valeur MOS vidéo
 * dans l'élément "vqmmos"
 *
 * <refsect2>
 * <title>Example launch line</title>
 * |[
 * gst-launch  --gst-debug=nrvqm_mosv:4,rtpsession_rtcp:5  playbin uri=rtsp://servername/filename.mp4
video-sink="nrvqm ! xvimagesink"
 * ]| Cette Pipeline affiche la vidéo filename.mp4 du serveur servename et calcule son MOS Vidéo et affiche sur
le terminal
 * les informations concernant les MOS Vidéo.
 * </refsect2>
 */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif

#include "gstnrvqm.h"
#include <string.h>
#include <math.h>
#include <gst/video/video.h>

// Définir catégorie de Debug
GST_DEBUG_CATEGORY_STATIC (gst_nrvqm_mosv);

GST_DEBUG_CATEGORY_STATIC (nrvqm_debug);
#define GST_CAT_DEFAULT nrvqm_debug

/* GstNrvqm signals and args */
enum
{
```

```
  /* FILL ME */
  LAST_SIGNAL
};

enum
{
  PROP_0,
  PROP_NRVQM
     /* FILL ME */
};

// Constantes
#define DEFAULT_PROP_NRVQM  1
#define DEFAULT_WINDOW_TIME  10000
#define DEFAULT_MOSV       95
#define DEFAULT_THRESHOLD    200


static const GstElementDetails nrvqm_details =
GST_ELEMENT_DETAILS ("Video nrvqm ",
   "Filter/Effect/Video",
   "Calcule le MOS/VQM d'un stream Vidéo",
   "Ronny Henkes ");

static GstStaticPadTemplate gst_nrvqm_src_template =
GST_STATIC_PAD_TEMPLATE ("src",
   GST_PAD_SRC,
   GST_PAD_ALWAYS,
   GST_STATIC_CAPS (GST_VIDEO_CAPS_YUV ("{ IYUV, I420, YV12 }"))
   );

static GstStaticPadTemplate gst_nrvqm_sink_template =
GST_STATIC_PAD_TEMPLATE ("sink",
   GST_PAD_SINK,
   GST_PAD_ALWAYS,
   GST_STATIC_CAPS (GST_VIDEO_CAPS_YUV ("{ IYUV, I420, YV12 }"))
   );

static void gst_nrvqm_set_property (GObject * object, guint prop_id,
   const GValue * value, GParamSpec * pspec);
static void gst_nrvqm_get_property (GObject * object, guint prop_id,
   GValue * value, GParamSpec * pspec);

static gboolean gst_nrvqm_set_caps (GstBaseTransform * base, GstCaps * incaps,
   GstCaps * outcaps);
static gboolean gst_nrvqm_sink_event(GstPad *pad, GstEvent * event);
static gboolean gst_nrvqm_src_event(GstPad *pad, GstEvent * event);
static int calc_ntg(guint64 duration);
static float gst_nrvqm_delta (GstNrvqm * nrvqm, GstBuffer * outbuf);
static int gst_nrvqm_freeze_detect (GstNrvqm *nrvqm, GstBuffer *outbuf);
static float gst_nrvqm_calc_vqm (GstNrvqm * nrvqm, GstBuffer * outbuf);
static GstFlowReturn gst_nrvqm_transform_ip (GstBaseTransform * transform,
   GstBuffer * buf);
static gboolean     gst_nrvqm_start       (GstBaseTransform *trans);
static gboolean     gst_nrvqm_stop        (GstBaseTransform *trans);
static gboolean     gst_sendmosv (GstNrvqm * nrvqm);
GST_BOILERPLATE (GstNrvqm, gst_nrvqm, GstVideoFilter, GST_TYPE_VIDEO_FILTER);
```

```
/* Initialisation de base */
static void
gst_nrvqm_base_init (gpointer g_class)
{
  GstElementClass *element_class = GST_ELEMENT_CLASS (g_class);

  gst_element_class_set_details (element_class, &nrvqm_details);

  gst_element_class_add_pad_template (element_class,
     gst_static_pad_template_get (&gst_nrvqm_sink_template));
  gst_element_class_add_pad_template (element_class,
     gst_static_pad_template_get (&gst_nrvqm_src_template));
}

/*Initialisation de la classe */
static void
gst_nrvqm_class_init (GstNrvqmClass * g_class)
{
  GObjectClass *gobject_class;
  GstBaseTransformClass *trans_class;



  gobject_class = G_OBJECT_CLASS (g_class);
  trans_class = GST_BASE_TRANSFORM_CLASS (g_class);

  gobject_class->set_property = gst_nrvqm_set_property;
  gobject_class->get_property = gst_nrvqm_get_property;

  g_object_class_install_property (gobject_class, PROP_NRVQM,
     g_param_spec_double ("nrvqm", "Nrvqm", "nrvqm",
        0.01, 10, DEFAULT_PROP_NRVQM, G_PARAM_READWRITE));

  trans_class->set_caps = GST_DEBUG_FUNCPTR (gst_nrvqm_set_caps);
  trans_class->transform_ip = GST_DEBUG_FUNCPTR (gst_nrvqm_transform_ip);
  trans_class->start = GST_DEBUG_FUNCPTR (gst_nrvqm_start); //RHE
  trans_class->stop = GST_DEBUG_FUNCPTR (gst_nrvqm_stop); //RHE


}


/* Traitement des événements interceptés sur le Sink Pad */
static gboolean
gst_nrvqm_sink_event(GstPad *pad, GstEvent * event)
 {
  GstNrvqm * nrvqm;
  gboolean ret;
  nrvqm=GST_NRVQM(gst_pad_get_parent (pad));

  switch (GST_EVENT_TYPE (event)) {
    case GST_EVENT_EOS:
      ret=gst_pad_push_event (GST_BASE_TRANSFORM_SRC_PAD(nrvqm),event);
      break;
    default:
      ret=gst_pad_push_event (GST_BASE_TRANSFORM_SRC_PAD(nrvqm),event);
```

```
    break;

  }
  gst_object_unref(nrvqm);
  return ret;

}

/*Traitement des événements interceptés sur le Source Pad */
static gboolean
gst_nrvqm_src_event(GstPad *pad, GstEvent * event)
 {
  GstNrvqm * nrvqm;
  gboolean ret;

  nrvqm=GST_NRVQM(gst_pad_get_parent (pad));

  switch (GST_EVENT_TYPE (event)) {
    case GST_EVENT_EOS:

      ret=gst_pad_push_event (GST_BASE_TRANSFORM_SINK_PAD(nrvqm),event);
      break;

    default:
      ret=gst_pad_push_event (GST_BASE_TRANSFORM_SINK_PAD(nrvqm),event);
      break;

  }
  gst_object_unref(nrvqm);
  return ret;

}

/* Initialisation de l'instance nrvqm*/
static void
gst_nrvqm_init (GstNrvqm * nrvqm, GstNrvqmClass * g_class)
{

  GST_DEBUG_OBJECT (nrvqm, "gst_nrvqm_init");

  /* initialisation des properties */
  nrvqm->nrvqm = DEFAULT_PROP_NRVQM;
  nrvqm->nbframes=0;
  nrvqm->lastbuf=gst_buffer_new();
  nrvqm->lasttime=0;
  nrvqm->testtime=0;
  nrvqm->nbdisc=0;
  nrvqm->quedisc=g_queue_new();
  /*assignation des fonctions de traitement des événements */
  gst_pad_set_event_function (GST_BASE_TRANSFORM_SINK_PAD(nrvqm),
     GST_DEBUG_FUNCPTR (gst_nrvqm_sink_event));
  gst_pad_set_event_function (GST_BASE_TRANSFORM_SRC_PAD(nrvqm),
     GST_DEBUG_FUNCPTR (gst_nrvqm_src_event));
  /*Initialisation de la catégorie de debugging*/
  GST_DEBUG_CATEGORY_INIT (gst_nrvqm_mosv, "nrvqm_mosv", 0, "NRVQM"); //RHE
}
```

```
/*Assignation des propriétés*/
static void
gst_nrvqm_set_property (GObject * object, guint prop_id, const GValue * value,
   GParamSpec * pspec)
{
 GstNrvqm *nrvqm;

 g_return_if_fail (GST_IS_NRVQM (object));
 nrvqm = GST_NRVQM (object);

 GST_DEBUG ("gst_nrvqm_set_property");
 switch (prop_id) {
  case PROP_NRVQM:
    nrvqm->nrvqm = g_value_get_double (value);

    break;
  default:
    G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
    break;
 }
}

/*Lecture des propriétés*/
static void
gst_nrvqm_get_property (GObject * object, guint prop_id, GValue * value,
   GParamSpec * pspec)
{
 GstNrvqm *nrvqm;

 g_return_if_fail (GST_IS_NRVQM (object));
 nrvqm = GST_NRVQM (object);

 switch (prop_id) {
  case PROP_NRVQM:
    g_value_set_double (value, nrvqm->nrvqm);
    break;
  default:
    G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
    break;
 }
}


// Fonction appellée quand l'élément NRVQM commence le traitement
static gboolean
gst_nrvqm_start       (GstBaseTransform *trans)
{
 /* démarrer un timer qui appelle la fonction gst_sendmosv chaque 400ms */
 GstNrvqm * nrvqm;
 nrvqm=GST_NRVQM(trans);
 g_timeout_add (400, (GSourceFunc) gst_sendmosv, nrvqm);

 // RHE TODO REININT VARS OPEN Files
 return TRUE;
};

// Fonction appellée quand l'élément NRVQM arrête le traitement
```

```
static gboolean
gst_nrvqm_stop       (GstBaseTransform *trans)
{

   disc_t *tmpdisc;
   GstNrvqm *nrvqm;
   nrvqm=GST_NRVQM(trans);
   /*libérer la mémoire */
    for (tmpdisc=g_queue_pop_head(nrvqm->quedisc) ; tmpdisc;  tmpdisc= g_queue_pop_head(nrvqm-
>quedisc))
     {

       g_free(tmpdisc);

    }
  nrvqm->nbdisc=0;
  //RHE TODO Close Files

  return TRUE;
};
/*static gboolean gst_sendmosv (GstNrvqm * nrvqm)
 *Cette fonction envoie le MOSV calculé sur le BUS GSTREAMER
 *
 *   @param nrvqm élément NRVQM
 *   @retourne toujours TRUE
 */
static gboolean
gst_sendmosv (GstNrvqm * nrvqm)
{
  float f_resvqm;
  GstEvent * vqmevent;
  GstStructure * vqmstructure;
  f_resvqm=0;
  //Appeller la fonction de calcul du MOSV

  f_resvqm=gst_nrvqm_calc_vqm (nrvqm, NULL);
  GST_CAT_INFO_OBJECT (gst_nrvqm_mosv,nrvqm, "Debug NRVQM Sending MOSV : %f QUEUE : %d TOTAL
DISC : %d",f_resvqm,g_queue_get_length(nrvqm->quedisc),nrvqm->nbdisc);

  //Créer et Envoyer un évenement sur le BUS GSTREAMER avec la valeur MOSV
  vqmstructure=gst_structure_new ("video/vqm","vqmmos", G_TYPE_DOUBLE,f_resvqm,NULL);
  vqmevent=gst_event_new_custom(GST_EVENT_CUSTOM_UPSTREAM ,vqmstructure);
  gst_pad_push_event(GST_BASE_TRANSFORM_SINK_PAD(nrvqm),vqmevent);


  return TRUE;
}


/* Useful macros */
#define GST_VIDEO_I420_Y_ROWSTRIDE(width) (GST_ROUND_UP_4(width))
#define GST_VIDEO_I420_U_ROWSTRIDE(width) (GST_ROUND_UP_8(width)/2)
#define GST_VIDEO_I420_V_ROWSTRIDE(width)
((GST_ROUND_UP_8(GST_VIDEO_I420_Y_ROWSTRIDE(width)))/2)

#define GST_VIDEO_I420_Y_OFFSET(w,h) (0)
```

```
#define GST_VIDEO_I420_U_OFFSET(w,h)
(GST_VIDEO_I420_Y_OFFSET(w,h)+(GST_VIDEO_I420_Y_ROWSTRIDE(w)*GST_ROUND_UP_2(h)))
#define GST_VIDEO_I420_V_OFFSET(w,h)
(GST_VIDEO_I420_U_OFFSET(w,h)+(GST_VIDEO_I420_U_ROWSTRIDE(w)*GST_ROUND_UP_2(h)/2))
#define GST_VIDEO_I420_SIZE(w,h)
(GST_VIDEO_I420_V_OFFSET(w,h)+(GST_VIDEO_I420_V_ROWSTRIDE(w)*GST_ROUND_UP_2(h)/2))

/* Assigne la capacités de l'élément NRVQM*/
static gboolean
gst_nrvqm_set_caps (GstBaseTransform * base, GstCaps * incaps,
   GstCaps * outcaps)
{
 GstNrvqm *this;
 GstStructure *structure;
 gboolean res;

 this = GST_NRVQM (base);

 GST_DEBUG_OBJECT (this,
    "set_caps: in %" GST_PTR_FORMAT " out %" GST_PTR_FORMAT, incaps, outcaps);

 structure = gst_caps_get_structure (incaps, 0);

 res = gst_structure_get_int (structure, "width", &this->width);
 res &= gst_structure_get_int (structure, "height", &this->height);
 if (!res)
  goto done;

 this->size = GST_VIDEO_I420_SIZE (this->width, this->height);

done:
 return res;
}


/*static int calc_ntg(guint64 duration)
 *calcule la classe de qualité en fonction du temps.
 *définition des classes de qualité pour un gel isolé de durée x :
 *x<71 ms classe 0 (excellent), 71<=x<532 ms classe 1 (Good), 532<=x<3495 ms classe 2(Fair)
 *3495<=x Classe 3 (Poor).
 *
 * @param duration temps en ms
 * @retourne la classe correspondante au temps fournis
 */

static int
calc_ntg(guint64 duration)
{

 int i_res = 0;
 if (duration < 71) i_res = 0; else if (duration < 532) i_res = 1;
  else if (duration < 3495) i_res = 2;  else if (duration > 3495) i_res = 3;


return i_res;
}
```

```
/* static float gst_nrvqm_delta (GstNrvqm * nrvqm, GstBuffer * outbuf)
 * Calcul du Delta entre l'image courante et précédente si celle existe,
 * sinon renvoie 0.
 *
 * @param nrvqm élément NRVQM
 * @param outbuf le buffer GST contenant l'image vidéo courante
 * @retourne la valeur du delta ou 0 si l'iamge précédente n'existe pas
 */
static float
gst_nrvqm_delta (GstNrvqm * nrvqm, GstBuffer * outbuf)

 {
  float d_delta;
  guint8  *data,*lastdata, *p_in,*p_last;
  guint size,iysize;
  int  i_comp_res, i_is_same;
  int i_pix, i_val_peak,i_diff_tmp;
  long l_comp_res;

  /*****************************************************/
  /* Comparer image i et image i-1 et Calculer delta */
  /*****************************************************/
  data = GST_BUFFER_DATA (outbuf);
  size = GST_BUFFER_SIZE (outbuf);
  if (nrvqm->lastbuf == NULL ) return 0;
  lastdata=GST_BUFFER_DATA (nrvqm->lastbuf);



  //Taille des données de la luminance, seul la luminance (Y) est pris en considération
  iysize=nrvqm->height * GST_VIDEO_I420_Y_ROWSTRIDE (nrvqm->width);


  //assigner les pointeurs de pixel
  p_in = data;
  p_last = lastdata;
  //Init vars: line numbers, visible pitch //
  i_val_peak = 0;
  i_comp_res = 0;
  i_diff_tmp = 0;
  //comparerer pixels et calculer
  for(i_pix=0;i_pix < iysize; i_pix++)
    {
     l_comp_res += (long) pow ((*p_in - *p_last),2) ;
     if (*p_in > i_val_peak) i_val_peak = *p_in;
     p_in++; p_last++;

  }


  // S'il n'y a pas de différence entre les images ils sont identiques
  if (l_comp_res == 0 ) {i_is_same = 1;};

  // Calcul du delta = f*[RMSE(I,I+1) / valpeak]
  d_delta = 100 * sqrt((float)  l_comp_res / (float)iysize) / i_val_peak ;

 return d_delta;
```

```
}


/* static int gst_nrvqm_freeze_detect (GstNrvqm *nrvqm, GstBuffer *outbuf)
 * détecter une discontinuité de fluidité c'est à dire un "gélé" d'image (freeze)
 * et l'insérer dans la queue de discontinuités si elle est > threshold.
 *
 * @param nrvqm élément NRVQM
 * @param outbuf le buffer GST contenant l'image vidéo courante
 * @retourne 1 s'il y a une discontinuité > threshold sinon 0
 * @ajoute les informations de la discontinuité dans une queue des
 * discontinuités si >threshold (timestamp, duration, mediatime)
 */
static int
gst_nrvqm_freeze_detect (GstNrvqm *nrvqm, GstBuffer *outbuf)
 {
  int i_res=0;
  int i_threshold=DEFAULT_THRESHOLD;
  disc_t * discont;
  guint64 courtime,difftime, timestamp, basetime, playtime,duration;

  basetime =GST_TIME_AS_MSECONDS(gst_element_get_base_time(GST_ELEMENT_CAST(nrvqm)));
  courtime=GST_TIME_AS_MSECONDS(gst_clock_get_time(GST_ELEMENT_CLOCK(nrvqm)));
  playtime=courtime-basetime;
  difftime=courtime - nrvqm->lasttime;
  //Calcul durée de ladiscontinuité et l'insérer dans la table exclure les premiers 700 ms
  if (outbuf != NULL && nrvqm->lasttime!=0 && GST_TIME_AS_MSECONDS(outbuf->timestamp) > 700)
   {

    timestamp=GST_TIME_AS_MSECONDS(outbuf->timestamp);

    /* Si duration > seuil détection,  insérer discontinuité dans la queue des discontinuités*/

    duration=abs(difftime);
    if ((duration > i_threshold) && (duration < DEFAULT_WINDOW_TIME) )
     {

      discont=g_new(disc_t,1);
      discont->timestamp=nrvqm->lasttimestamp;
      discont->duration=duration;
      discont->mediatime=(playtime - duration);
      g_queue_push_head (nrvqm->quedisc, discont);
      nrvqm->nbdisc++;
      i_res=1;
      printf("  Discontinuité : DiscPlaytime %lld Duration %lld, courtime %lld \n",discont->mediatime, duration,
courtime );
      GST_CAT_INFO_OBJECT (gst_nrvqm_mosv,nrvqm, "Duration : %lld  ",duration);
    }

  }
  return i_res;

}

/* static float gst_nrvqm_calc_vqm (GstNrvqm * nrvqm, GstBuffer * outbuf)
 * Calculer le MOS Vidéo en se basant sur la queue des discontinuités
 * et d'une discontinuité éventuelle non encore détectée.
```

```
 * Supprimer les discontinuités de la queue dont le temps de fin de la discontinuité
 * remonte à plus de DEFAULT_WINDOW_TIME.
 *
 * @param nrvqm élément NRVQM
 * @param outbuf le buffer GST contenant l'image vidéo courante
 * @retourne le MOS Vidéo
 */
static float
gst_nrvqm_calc_vqm (GstNrvqm * nrvqm, GstBuffer * outbuf)
 {

  guint64 courtime, basetime, playtime;
  disc_t   *tmpdisc;
  int i_dmax,i_ntg, i_tab_ntg[20],i;
  float f_dtotal, f_dpooling, f_dpool_tmp,  f_etgi, f_pntg, f_s_exp, f_r_exp, f_par_c, f_par_b;
  float f_pmax, f_pmin,f_mosref, f_mmax, f_mmin, f_res;
  f_mosref = DEFAULT_MOSV; i_dmax = 90; f_s_exp = 1.01; f_r_exp = 1.5;
  f_par_c = 27;f_par_b = 562;f_mmax = 85.8; f_mmin = 32.77;
  f_pmax = 2.017;f_pmin =1.1131;
  f_res=0;
  basetime =GST_TIME_AS_MSECONDS(gst_element_get_base_time(GST_ELEMENT_CAST(nrvqm)));
  courtime=GST_TIME_AS_MSECONDS(gst_clock_get_time(GST_ELEMENT_CLOCK(nrvqm)));
  playtime=courtime-basetime;

   for ( i = 0; i < 4; i++ ) i_tab_ntg[i] = 0;

   /* Supprimer les discontinuités de la queue dont le temps de fin de la discontinuité
      est plus grand que DEFAULT_WINDOW_TIME et calculer la table ntg  */
    for ( i=g_queue_get_length(nrvqm->quedisc); i>0; i--)
      {
       tmpdisc= g_queue_peek_nth(nrvqm->quedisc,i-1);

            if ((playtime - tmpdisc->mediatime ) > (DEFAULT_WINDOW_TIME + tmpdisc->duration))
        {

         tmpdisc=g_queue_pop_nth(nrvqm->quedisc,i-1);
         g_free(tmpdisc);
        }
        else
         {
         // Calculer n(tg) pour la durée de la discontinuité
         i_tab_ntg[calc_ntg(tmpdisc->duration)]++;
        }
    }


   /* Calculer MOSV */
   f_dpool_tmp = 0;
   // Tenir compte d'une discontinuité non encore détecté prendre courtime -nrvqm->lasttime comme temps
de freeze

 if ((nrvqm->lasttime != 0) && ((courtime -nrvqm->lasttime)> DEFAULT_THRESHOLD))
    {

     i_tab_ntg[calc_ntg(courtime -nrvqm->lasttime)]++;

     i_ntg = i_tab_ntg[calc_ntg(courtime -nrvqm->lasttime)];
```

```
    f_pntg = f_pmax - ((f_pmax-f_pmin)/ (1 + pow ((f_par_c/ i_ntg),f_r_exp)));
    f_etgi =      f_mosref -  f_mmax
                +((f_mmax-f_mmin)/ (1 + pow (f_par_b / (courtime -nrvqm->lasttime),f_s_exp)));
    f_dpool_tmp += pow(f_etgi,(f_pntg));
    printf(" in disc courtime -nrvqm->lasttime %lld \n", (courtime -nrvqm->lasttime));
  }


  // Calculer dpooling² de la queue
  for ( i=g_queue_get_length(nrvqm->quedisc); i>0; i--)

   {
     tmpdisc=g_queue_peek_nth(nrvqm->quedisc,i-1);
     i_ntg = i_tab_ntg[calc_ntg(tmpdisc->duration)];
     f_pntg = f_pmax - ((f_pmax-f_pmin)/ (1 + powf ((f_par_c/ i_ntg),f_r_exp)));
     f_etgi =      f_mosref -  f_mmax
                +((f_mmax-f_mmin)/ (1 + pow (f_par_b / tmpdisc->duration,f_s_exp)));
     f_dpool_tmp += pow(f_etgi,(f_pntg));

   }



  //Calculer dpooling
  f_dpooling = sqrt(f_dpool_tmp);
  //Calculer degradation total, cast float to int
  if (f_dpooling < i_dmax ) f_dtotal =  f_dpooling; else f_dtotal = i_dmax;
  // Calculer MOSV total
  f_res = (f_mosref -  f_dtotal);
  if (f_res < 10 ) f_res=10;
   printf(" Time %lld, VQM %f , courtime %lld \n", playtime,f_res,courtime);
 return f_res;
}


/* static GstFlowReturn gst_nrvqm_transform_ip (GstBaseTransform * base, GstBuffer * outbuf)
 * Cette fonction est appelée chaque fois qu'un nouveau GstBuffer est disponible pour
 * l'élément NRVQM.
 * On appelle la détection des discontinuités, met à jour
 * les propriétés nrvqm->lasttime et nrvqm->lasttimestamp
 * et on copie l'image courante vers nrvqm->lastbuf
 *
 * @param base élément GstBaseTransfom
 * @param outbuf le buffer GST contenant l'image vidéo courante
 * @retourne GST_FLOW_ERROR si la taille du buffer n'est pas correcte sinon GST_FLOW_OK
 */
static GstFlowReturn
gst_nrvqm_transform_ip (GstBaseTransform * base, GstBuffer * outbuf)
{
 GstNrvqm *nrvqm;
 guint8 *data;
 guint size;
 GstClockTime tmptime;
 GstClock * nrvqmclock;

 nrvqm = GST_NRVQM (base);
```

```
 data = GST_BUFFER_DATA (outbuf);
 size = GST_BUFFER_SIZE (outbuf);

 if (size != nrvqm->size)
  {
  GST_ELEMENT_ERROR (nrvqm, STREAM, FORMAT,
     (NULL), ("Invalid buffer size %d, expected %f", size, nrvqm->size));
  return GST_FLOW_ERROR;
 }
 nrvqmclock=GST_ELEMENT_CLOCK(nrvqm);
 // On saute la première image
 if (GST_BUFFER_SIZE(nrvqm->lastbuf) > 0)
   {
    //Appel de la détection des discontinuités
    gst_nrvqm_freeze_detect (nrvqm, outbuf);
    tmptime=gst_clock_get_time(nrvqmclock);
    nrvqm->lasttime=GST_TIME_AS_MSECONDS(tmptime);
  }

 nrvqm->lasttimestamp=GST_TIME_AS_MSECONDS(outbuf->timestamp);
 gst_buffer_set_data(nrvqm->lastbuf, data, size);

 return GST_FLOW_OK;
}



/* Initialisation du plugin*/

static gboolean
plugin_init (GstPlugin * plugin)
{
 GST_DEBUG_CATEGORY_INIT (nrvqm_debug, "nrvqm", 0, "nrvqm");

 return gst_element_register (plugin, "nrvqm", GST_RANK_NONE, GST_TYPE_NRVQM);
}

GST_PLUGIN_DEFINE (GST_VERSION_MAJOR,
   GST_VERSION_MINOR,
   "nrvqm",
   "Changes nrvqm on video images",
   plugin_init, VERSION, GST_LICENSE, GST_PACKAGE_NAME, GST_PACKAGE_ORIGIN);
```

B.2 Le fichier Source gstnrvqm.h

```
/* GStreamer
 * Copyright (C) <1999> Erik Walthinsen <omega@cse.ogi.edu>
 * Copyright (C) <2003> David Schleef <ds@schleef.org>
 * Copyright (C) 2003 Arwed v. Merkatz <v.merkatz@gmx.net>
 * Copyright (C) 2006 Mark Nauwelaerts <manauw@skynet.be>
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Library General Public
 * License as published by the Free Software Foundation; either
 * version 2 of the License, or (at your option) any later version.
```

```
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Library General Public License for more details.
 *
 * You should have received a copy of the GNU Library General Public
 * License along with this library; if not, write to the
 * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
 * Boston, MA 02111-1307, USA.
 */


#ifndef __GST_VIDEO_NRVQM_H__
#define __GST_VIDEO_NRVQM_H__

#include <gst/video/gstvideofilter.h>

G_BEGIN_DECLS

#define GST_TYPE_NRVQM \
  (gst_nrvqm_get_type())
#define GST_NRVQM(obj) \
  (G_TYPE_CHECK_INSTANCE_CAST((obj),GST_TYPE_NRVQM,GstNrvqm))
#define GST_NRVQM_CLASS(klass) \
  (G_TYPE_CHECK_CLASS_CAST((klass),GST_TYPE_NRVQM,GstNrvqmClass))
#define GST_IS_NRVQM(obj) \
  (G_TYPE_CHECK_INSTANCE_TYPE((obj),GST_TYPE_NRVQM))
#define GST_IS_NRVQM_CLASS(klass) \
  (G_TYPE_CHECK_CLASS_TYPE((klass),GST_TYPE_NRVQM))

typedef struct _GstNrvqm GstNrvqm;
typedef struct _GstNrvqmClass GstNrvqmClass;
typedef struct disc_t {
   guint64    timestamp; //timestamp de l'image
   guint64    mediatime;//temps de lecture Media en ms
   guint64    duration; //durée de la discontinuité en ms
} disc_t;

/**
 * GstNrvqm:
 *
 * Opaque data structure.
 */
struct _GstNrvqm
{
  GstVideoFilter videofilter;

  /* format */
  gint width;
  gint height;
  gint size;

  /* properties */
  double nrvqm;
  double nbframes;
  guint   nbdisc; // nombre total des discontinuités
```

```
  guint64 lasttime;//Temps de lecture de la derniere image
  guint64 lasttimestamp;// Timestamp de la derniere Image
  guint64 testtime;
  GQueue * quedisc;
  GstBuffer * lastbuf;
  float lastdelta;
  /* tables */
  guint8 nrvqm_table[256];
};

struct _GstNrvqmClass
{
  GstVideoFilterClass parent_class;
};



GType gst_nrvqm_get_type(void);

G_END_DECLS

#endif /* __GST_VIDEO_NRVQM_H__ */
```

Annexe C

L'annexe C reprend le code source de plusieurs fonctions ajoutées ou adaptées aux éléments GstRTSPSrc, GstRtpBin, GstRtpSession, GstRtcpBuffer et RtpSession existants. Seules les fonctions principales on été repris. Les fichiers avec le code source complet se trouvent sur le CD d'accompagnement.
Pour faciliter la lecture, les parties du code modifié, ont été mis en italique/ gras.

Les fichiers relatifs à GstRTSPSrc sont gstrtspsrc.c et.h, ils appartiennent au plugin RTSP qui est compris dans la compilation de plugin GStreamer, 'plugins-good 0.10.16'.

Les fichiers relatifs à GstRtpBin sont gstrtpbin.c et.h, ils appartiennent au plugin rtpmanager qui est compris dans la compilation de plugin GStreamer, 'plugins-good 0.10.16'.

Les fichiers relatifs à GstRtpSession sont gstrtpsession.c et.h, ils appartiennent au plugin rtpmanager qui est compris dans la compilation de plugin GStreamer, 'plugins-good 0.10.16'.

Les fichiers relatifs à RtpSession sont rtpsession.c et.h, ils appartiennent au plugin rtpmanager qui est compris dans la compilation de plugin GStreamer, 'plugins-good 0.10.16'.

Les fichiers relatifs à GstRtcpbuffer sont gstrtcpbuffer.c et.h, ils appartiennent au plugin rtp qui est compris dans la compilation de plugin GStreamer, 'plugins-base 0.10.25'.

C.1 gst_rtspsrc_handle_src_event(…)

Cette fonction et contenu dans le fichier gstrtspsrc.c

```
static gboolean
gst_rtspsrc_handle_src_event (GstPad * pad, GstEvent * event)
{
  GstRTSPSrc *src;
  gboolean res = TRUE;
  gboolean forward;
  gboolean b_res;
  GstStructure * structure;

  int vqmrequest;

  src = GST_RTSPSRC_CAST (gst_pad_get_parent (pad));

  GST_DEBUG_OBJECT (src, "pad %s:%s received event %s",
     GST_DEBUG_PAD_NAME (pad), GST_EVENT_TYPE_NAME (event));

  switch (GST_EVENT_TYPE (event)) {
    case GST_EVENT_SEEK:
      res = gst_rtspsrc_perform_seek (src, event);
      forward = FALSE;
      break;

    // Ajouté par Ronny Henkes Evenement vqmmos
    case GST_EVENT_CUSTOM_UPSTREAM:
    forward = TRUE;
    structure = (GstStructure *)gst_event_get_structure(event);
    b_res =  gst_structure_get_int (structure, "vqmmos", &vqmrequest);
    /* RHE - Si lévènement contient un élément "vqmmos"
     * transmettre la valeur MOS Vidéo au RTP Session Manager
     */
     if (b_res )
       {
        forward = FALSE;
        g_object_set (src->session, "imosv", vqmrequest, NULL);
       }
    break;


    case GST_EVENT_QOS:
    case GST_EVENT_NAVIGATION:
    case GST_EVENT_LATENCY:

    default:
      forward = TRUE;
      break;
  }
  if (forward) {
    GstPad *target;

    if ((target = gst_ghost_pad_get_target (GST_GHOST_PAD_CAST (pad)))) {
      res = gst_pad_send_event (target, event);
```

```
      gst_object_unref (target);
    } else {
      gst_event_unref (event);
    }
  } else {
    gst_event_unref (event);
  }
  gst_object_unref (src);

  return res;
}
```

C.2 gst_rtp_bin_set_property(…)

Cette fonction et contenu dans le fichier gstrtpbin.c

```
static void
gst_rtp_bin_set_property (GObject * object, guint prop_id,
    const GValue * value, GParamSpec * pspec)
{
  GstRtpBin *rtpbin;

  rtpbin = GST_RTP_BIN (object);

  switch (prop_id) {
    case PROP_LATENCY:
      GST_RTP_BIN_LOCK (rtpbin);
      rtpbin->latency = g_value_get_uint (value);
      GST_RTP_BIN_UNLOCK (rtpbin);
      /* propegate the property down to the jitterbuffer */
      gst_rtp_bin_propagate_property_to_jitterbuffer (rtpbin, "latency", value);
      break;
    // RHE -Ajouté par Henkes Ronny -
    // Propager la Valeur MOS Video vers les sessions RTP
    case PROP_IMOSV:
      GST_RTP_BIN_LOCK (rtpbin);
      GST_RTP_BIN_UNLOCK (rtpbin);
      gst_rtp_bin_propagate_property_to_rtpsession (rtpbin, "imosv", value);
      break;
    case PROP_SDES:
      gst_rtp_bin_set_sdes_struct (rtpbin, g_value_get_boxed (value));
      break;
    case PROP_DO_LOST:
      GST_RTP_BIN_LOCK (rtpbin);
      rtpbin->do_lost = g_value_get_boolean (value);
      GST_RTP_BIN_UNLOCK (rtpbin);
      gst_rtp_bin_propagate_property_to_jitterbuffer (rtpbin, "do-lost", value);
      break;
    case PROP_IGNORE_PT:
      rtpbin->ignore_pt = g_value_get_boolean (value);
      break;
    default:
      G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
      break;
  }
}
```

C.3 gst_rtp_bin_propagate_property_to_rtpsession(…)

Cette fonction se trouve dans le fichier gstrtpbin.c.

```
/*RHE - Propager une propriété vers les Sessions
 * RTP (GstRtpSession)
 *
 */
static void
gst_rtp_bin_propagate_property_to_rtpsession (GstRtpBin * bin,
    const gchar * name, const GValue * value)
{
  GSList *sessions;
  GstRtpSession * sess;

  GST_RTP_BIN_LOCK (bin);
  for (sessions = bin->sessions; sessions; sessions = g_slist_next (sessions)) {
    GstRtpBinSession *session = (GstRtpBinSession *) sessions->data;

    GST_RTP_SESSION_LOCK (session);

      sess = (GstRtpSession *) session->session;

      g_object_set_property (G_OBJECT (sess), name, value);

    GST_RTP_SESSION_UNLOCK (session);
  }
  GST_RTP_BIN_UNLOCK (bin);
}
```

C.4 gst_rtp_session_set_property(…)

Cette fonction se trouve dans le fichier gstrtpsession.c.

```
static void
gst_rtp_session_set_property (GObject * object, guint prop_id,
    const GValue * value, GParamSpec * pspec)
{
  GstRtpSession *rtpsession;
  GstRtpSessionPrivate *priv;
  rtpsession = GST_RTP_SESSION (object);
  priv = rtpsession->priv;
  switch (prop_id) {
    case PROP_NTP_NS_BASE:
      GST_OBJECT_LOCK (rtpsession);
      priv->ntpnsbase = g_value_get_uint64 (value);
      GST_DEBUG_OBJECT (rtpsession, "setting NTP base to %" GST_TIME_FORMAT,
          GST_TIME_ARGS (priv->ntpnsbase));
      GST_OBJECT_UNLOCK (rtpsession);
```

```
      break;
    case PROP_BANDWIDTH:
      rtp_session_set_bandwidth (priv->session, g_value_get_double (value));
      break;
    case PROP_RTCP_FRACTION:
      rtp_session_set_rtcp_fraction (priv->session, g_value_get_double (value));
      break;
    case PROP_SDES:
      rtp_session_set_sdes_struct (priv->session, g_value_get_boxed (value));
      break;
    // RHE - Assigner la variable i_mosv de la Session RTP (RtpSession) avec MOSV
    case PROP_IMOSV:
      rtp_session_set_imosv(priv->session, g_value_get_float(value));
      break;
    default:
      G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
      break;
  }
}
```

C.5 gst_rtp_session_set_mosv(…)

Cette fonction se trouve dans le fichier gstrtpsession.c.

```
//RHE SET MOSV on RTPSESSION
void
gst_rtp_session_set_mosv (GstElement *element, gfloat mosv)
{
  GstRtpSession * session;
  session= (GstRtpSession *) element;
  session->priv->session->i_mosv=mosv;


}
```

C.6 La structure rtpsession

La structure rtpsession est définie das le fichier rtpsession.h

```
/**
 * RTPSession:
 * @lock: lock to protect the session
 * @source: the source of this session
 * @ssrcs: Hashtable of sources indexed by SSRC
 * @cnames: Hashtable of sources indexed by CNAME
 * @num_sources: the number of sources
 * @activecount: the number of active sources
 * @callbacks: callbacks
 * @user_data: user data passed in callbacks
 * @stats: session statistics
 * @conflicting_addresses: GList of conflicting addresses
```

```
 *
 * The RTP session manager object
 */
struct _RTPSession {
  GObject      object;
  GMutex       *lock;
  guint        header_len;
  guint        mtu;
  RTPSource    *source;
  /* Metrique MOS Video RHE */
  gfloat       i_mosv;
  /* for sender/receiver counting */
  guint32      key;
  guint32      mask_idx;
  guint32      mask;
  GHashTable   *ssrcs[32];
  GHashTable   *cnames;
  guint        total_sources;
  GstClockTime  next_rtcp_check_time;
  GstClockTime  last_rtcp_send_time;
  gboolean     first_rtcp;
  gchar        *bye_reason;
  gboolean     sent_bye;
  RTPSessionCallbacks   callbacks;
  gpointer             process_rtp_user_data;
  gpointer             send_rtp_user_data;
  gpointer             send_rtcp_user_data;
  gpointer             sync_rtcp_user_data;
  gpointer             clock_rate_user_data;
  gpointer             reconsider_user_data;
  RTPSessionStats stats;
  GList        *conflicting_addresses;
  gboolean     change_ssrc;
};
```

C.7 gst_rtcp_packet_add_xr(…)

Cette function se trouve dans le fichier rtpsession.c.

```
/** RHE
 * gst_rtcp_packet_add_xr:
 * @packet: a valid  #GstRTCPPacket
 * @ssrc: data source being reported
 * Add a new RTCP XR QoE Infos to @packet with the given values.
 * Returns: %TRUE if the packet was created. This function can return %FALSE if
 * the max MTU is exceeded or the number of report blocks is greater than
 * #GST_RTCP_MAX_RB_COUNT.
 */
gboolean
gst_rtcp_packet_add_xr (GstRTCPPacket * packet, guint32 ssrc,guint32 name, gfloat vqmmos)
{
```

```
  guint8 *data;
  guint size, offset;
  guint16 i_vqmmos;
  guint8 i_decmos;
  gfloat f_tmp;
//Calcul de la représentation en scaled integer 8:8
  i_vqmmos= (int) vqmmos;
  f_tmp = vqmmos - i_vqmmos;
  i_decmos = (int) (256 * f_tmp);
  g_return_val_if_fail (packet != NULL, FALSE);
  g_return_val_if_fail (GST_IS_BUFFER (packet->buffer), FALSE);

  if (packet->count >= GST_RTCP_MAX_RB_COUNT)
    goto no_space;

  data = GST_BUFFER_DATA (packet->buffer);
  size = GST_BUFFER_SIZE (packet->buffer);

  // skip header
  offset = packet->offset + 4;

  /*if (packet->type == GST_RTCP_TYPE_RR)
    offset += 4;
  else
    offset += 24;*/

  // move to current index
  offset += (packet->count * 24);

  // we need 24 free bytes now
  if (offset + 12 >= size)
    goto no_space;

  // increment packet count and length
  packet->count++;
  data[packet->offset]++;
  packet->length += 3;
  data[packet->offset + 2] = (packet->length) >> 8;
  data[packet->offset + 3] = (packet->length) & 0xff;
  // move to new report block offset
//Ajouter les données dans le paquet
  data += offset;
  GST_WRITE_UINT32_BE (data, ssrc);
  data += 4;
  GST_WRITE_UINT8 (data, 100);//BT
  data += 1;
  GST_WRITE_UINT8 (data, 0); //dir=0; Tag=0
  data += 1;
  GST_WRITE_UINT16_BE (data,1); //Pour l'instant 1 Seul SSRC
  data += 2;
  GST_WRITE_UINT16_BE (data, 516);//channel =0; dir = 0, type = '0100' : MOSV, calc alg = 4
  data += 2;
  GST_WRITE_UINT8 (data,i_vqmmos );// Octet 1 : partie entière
  data += 1;
  GST_WRITE_UINT8 (data, i_decmos); //Octet 2: partie décimale
  data += 1;
  return TRUE;
```

```
no_space:
 {
  return FALSE;
 }
}
```

C.8 rtp_session_on_timeout(…)

Cette fonction se trouve dans le fichier rtpsession.c.

```
/**
 * rtp_session_on_timeout:
 * @sess: an #RTPSession
 * @current_time: the current system time
 * @ntpnstime: the current NTP time in nanoseconds
 *
 * Perform maintenance actions after the timeout obtained with
 * rtp_session_next_timeout() expired.
 *
 * This function will perform timeouts of receivers and senders, send a BYE
 * packet or generate RTCP packets with current session stats.
 *
 * This function can call the #RTPSessionSendRTCP callback, possibly multiple
 * times, for each packet that should be processed.
 *
 * Returns: a #GstFlowReturn.
 */
GstFlowReturn
rtp_session_on_timeout (RTPSession * sess, GstClockTime current_time,
    guint64 ntpnstime)
{
  GstFlowReturn result = GST_FLOW_OK;
  GList *item;
  ReportData data;
  RTPSource *own;
  gboolean notify = FALSE;
  GstRTCPPacket rtcpapp;
  g_return_val_if_fail (RTP_IS_SESSION (sess), GST_FLOW_ERROR);

  GST_DEBUG ("reporting at %" GST_TIME_FORMAT ", NTP time %" GST_TIME_FORMAT,
      GST_TIME_ARGS (current_time), GST_TIME_ARGS (ntpnstime));
  data.sess = sess;
  data.rtcp = NULL;
  data.current_time = current_time;
  data.ntpnstime = ntpnstime;
  data.is_bye = FALSE;
  data.has_sdes = FALSE;
  own = sess->source;
  RTP_SESSION_LOCK (sess);
  /* get a new interval, we need this for various cleanups etc */
  data.interval = calculate_rtcp_interval (sess, TRUE, sess->first_rtcp);

  /* first perform cleanups */
  g_hash_table_foreach_remove (sess->ssrcs[sess->mask_idx],
      (GHRFunc) session_cleanup, &data);
```

```
 /* see if we need to generate SR or RR packets */
 if (is_rtcp_time (sess, current_time, &data)) {
  if (own->received_bye) {
   /* generate BYE instead */
   GST_DEBUG ("generating BYE message");
   session_bye (sess, &data);
   sess->sent_bye = TRUE;
  } else {

   /* loop over all known sources and do something */
   g_hash_table_foreach (sess->ssrcs[sess->mask_idx],
      (GHFunc) session_report_blocks, &data);

  }
 }

 if (data.rtcp) {
  guint size;

  /* we keep track of the last report time in order to timeout inactive
   * receivers or senders */
  sess->last_rtcp_send_time = data.current_time;
  sess->first_rtcp = FALSE;

  /* add SDES for this source when not already added */
  if (!data.has_sdes)
   session_sdes (sess, &data);

  //RHE Ajout VQM/MOS Report Block à la fin si on a reçu une valeur MOS calculée
  if((gst_rtcp_packet_get_count (&data.packet)>0) && (sess->i_mosv >0))
   {
    GST_CAT_INFO_OBJECT (rtp_session_debug_rtcp,sess, "Debug RTPSession transmit RTCP MOSV :
%f",sess->i_mosv);
    gst_rtcp_buffer_add_packet(data.rtcp,GST_RTCP_TYPE_XR, &rtcpapp); //207:RTCP XR paquet
    gst_rtcp_packet_add_xr (&rtcpapp,sess->source->ssrc,1297044310,sess->i_mosv);//1297044310 :
"MOSV"
  }

  /* update average RTCP size before sending */
  size = GST_BUFFER_SIZE (data.rtcp) + sess->header_len;
  UPDATE_AVG (sess->stats.avg_rtcp_packet_size, size);
 }

 /* check for outdated collisions */
 GST_DEBUG ("checking collision list");
 item = g_list_first (sess->conflicting_addresses);
 while (item) {
  RTPConflictingAddress *known_conflict = item->data;
  GList *next_item = g_list_next (item);

  if (known_conflict->time < current_time - (data.interval *
       RTCP_INTERVAL_COLLISION_TIMEOUT)) {
   sess->conflicting_addresses =
     g_list_delete_link (sess->conflicting_addresses, item);
   GST_DEBUG ("collision %p timed out", known_conflict);
   g_free (known_conflict);
  }
```

```
    item = next_item;
  }

  if (sess->change_ssrc) {
    GST_DEBUG ("need to change our SSRC (%08x)", own->ssrc);
    g_hash_table_steal (sess->ssrcs[sess->mask_idx],
        GINT_TO_POINTER (own->ssrc));

    own->ssrc = rtp_session_create_new_ssrc (sess);
    rtp_source_reset (own);

    g_hash_table_insert (sess->ssrcs[sess->mask_idx],
        GINT_TO_POINTER (own->ssrc), own);

    g_free (sess->bye_reason);
    sess->bye_reason = NULL;
    sess->sent_bye = FALSE;
    sess->change_ssrc = FALSE;
    notify = TRUE;
    GST_DEBUG ("changed our SSRC to %08x", own->ssrc);
  }
  RTP_SESSION_UNLOCK (sess);

  if (notify)
    g_object_notify (G_OBJECT (sess), "internal-ssrc");

  /* push out the RTCP packet */
  if (data.rtcp) {
    /* close the RTCP packet */
    gst_rtcp_buffer_end (data.rtcp);

    GST_DEBUG ("sending packet");
    if (sess->callbacks.send_rtcp)
      result = sess->callbacks.send_rtcp (sess, own, data.rtcp,
          sess->sent_bye, sess->send_rtcp_user_data);
    else {
      GST_DEBUG ("freeing packet");
      gst_buffer_unref (data.rtcp);
    }
  }

  return result;
}
```

C.9 Le GstRTCPType

Cette structure se trouve dans le fichier gstrtcpbuffer.h

```
/**
 * GstRTCPType:
 * @GST_RTCP_TYPE_INVALID: Invalid type
 * @GST_RTCP_TYPE_SR: Sender report
 * @GST_RTCP_TYPE_RR: Receiver report
 * @GST_RTCP_TYPE_SDES: Source description
 * @GST_RTCP_TYPE_BYE: Goodbye
 * @GST_RTCP_TYPE_APP: Application defined
 * @GST_RTCP_TYPE_RTPFB: Transport layer feedback. Since: 0.10.23
 * @GST_RTCP_TYPE_PSFB: Payload-specific feedback. Since: 0.10.23
```

```
 *
 * Different RTCP packet types.
 */
typedef enum
{
  GST_RTCP_TYPE_INVALID = 0,
  GST_RTCP_TYPE_SR      = 200,
  GST_RTCP_TYPE_RR      = 201,
  GST_RTCP_TYPE_SDES    = 202,
  GST_RTCP_TYPE_BYE     = 203,
  GST_RTCP_TYPE_APP     = 204,
  GST_RTCP_TYPE_RTPFB   = 205,
  GST_RTCP_TYPE_PSFB    = 206,
  GST_RTCP_TYPE_XR      = 207  // définition du  paquet RTCP de type XR
} GstRTCPType;
```

Annexe D

Cette annexe comprend des informations sur l'installation du plugin NRVQM.

CONFIGURATION REQUISE pour l'installation :

Le plus simple pour installer le plugin NRVQM est d'utiliser Linux Ubuntu 9.10, qui propose déjà les modules principaux.

(sinon il faut les modules suivants :

-gstreamer base 0.10.25
-GLib-2.22.4, and libxml2-2.7.6
-gstreamer plugins-base 0.10.25
-GStreamer-0.10.25, liboil-0.3.16, and Pango-1.26.2
-gstreamer plugins-good 0.10.16
-gstreamer-x 0.10.25 (X11))

en plus il faut installer les modules suivants (versions Karmic Koala, Ubuntu 9.10):

(sous Ubuntu)
sudo apt-get install gstreamer-tools
sudo apt-get install gstreamer0.10-ffmpeg
sudo apt-get install libtool

INSTALLATION NRVQM

(Pour l'installation on suppose qu'on utilise un ordinateur avec le système d'exploitation Ubuntu 9.10)

-copiez l'archive "installnrvqm.tar.gz" du cd d'accompagnement vers le répertoire "Home" de l'utilisateur courant (par exemple).
-extrayez l'archive dans ce répertoire

-ouvrez un terminal et allez dans le répertoire "home"/installnrvqm
-tapez sudo ./instlibs
-si des erreurs de répertoire sont affichées, il faut adapter les répertoires (voir plus loin)
-Vous pouvez vérifier si le plugin s'est bien installé en tappant : gst-inspect nrvqm.
Cette commande devrait vous afficher des informations sur le plugin
-pour afficher une vidéo et afficher les MOS dans un terminal on peut utiliser la commande :
gst-launch --gst-debug=nrvqm_mosv:4,rtpsession_rtcp:5 playbin uri=rtsp://adresseserveur/nomvideo.mp4 video-sink="nrvqm ! xvimagesink"

PROBLEMES LORS DE L'INSTALLATION
-répertoires incorrects lors de la commande sudo ./instlibs :
il faut adapter le répertoire de base des libs Gstreamer.
Par defaut en Ubuntu 9.10 ils se situent dans /usr/lib/* , en Ubuntu 8.10 dans /usr/local/lib/*
Il faut adapter les répertoires dans les fichies instlibs et les 5 fichiers *.la dans les sous-répertoires de installnrvqm.